10707
Deep Learning: Spring 2023

Russ Salakhutdinov

Machine Learning Department
rsalakhu@cs.cmu.edu

Lectures 1,2

Evaluation

- 3 Assignments, worth 60%.
- Mid-term Exam 10%
- Projects, 30%:
  - Midway report 5%, Final Project 25%.

Homework Dates – Check the website for updates!

Evaluation

- 5 late days for all assignments.
- No more than 3 late days per assignment. After 3 late days, you will get 0.

- 3 late days for projects: can be split between project proposal and final project.

- Project: Teams of 2 people per project.

Project

- The idea of the final project is to give you some experience trying to do a piece of original research in machine learning and coherently writing up your result.
- What is expected: A simple but original idea that you describe clearly, relate to existing methods, implement and test on some real-world problem.
- To do this you will need to write some basic code, run it on some data, make some figures, read a few background papers, collect some references, and write an 8-page report describing your model, algorithm, and results.

Text Books

- **Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016)**
Deep Learning Book (available online)

- **Christopher M. Bishop (2006) Pattern Recognition and Machine Learning, Springer.**

- Kevin Murphy (2013)
Machine Learning: A Probabilistic Perspective

- Trevor Hastie, Robert Tibshirani, Jerome Friedman (2009) The Elements of Statistical Learning (available online)

- David MacKay (2003) Information Theory, Inference, and Learning Algorithms

- Most of the figures and material will come from these books.

Online Resources

- I will be using a number of online resources, including

- Joan Bruna's Deep Learning Course
http://joanbruna.github.io/stat212b/

- Hugo Larochelle Neural Network Course
http://info.usherbrooke.ca/hlarochelle/neural_networks/description.html

- Deep Learning Summer School in Montreal
https://sites.google.com/site/deeplearningsummerschool2016/home

- I will be adding more resources, check the webpage.

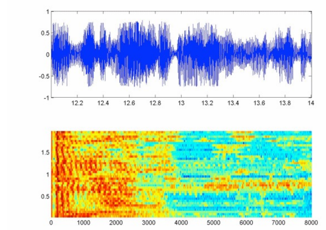
Mining for Structure

Massive increase in both computational power and the amount of data available from web, video cameras, laboratory measurements.

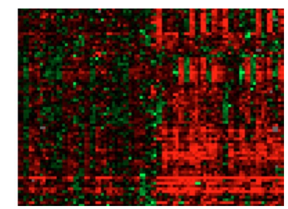
Images & Video

Text & Language

Speech & Audio

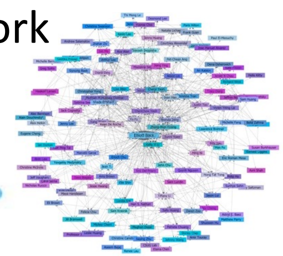


Gene Expression



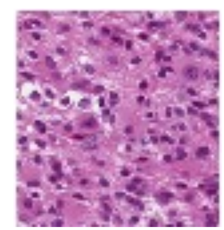
Product Recommendation

Relational Data/ Social Network



fMRI

Tumor region



- Develop statistical models that can discover underlying structure, cause, or statistical correlation from data.
- Multiple application domains.

Impact of Deep Learning

- Speech Recognition
- Computer Vision
- Recommender Systems
- Language Understanding
- Drug Discovery and Medical Image Analysis

Example: Boltzmann Machine

Latent (hidden) variables

Model parameters

$$P(\mathbf{x}, \mathbf{y}) = \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \exp\left[\mathbf{x}^{\top}\mathbf{W}^{(1)}\mathbf{h} + \mathbf{y}^{\top}\mathbf{W}^{(2)}\mathbf{h}\right]$$

Input data (e.g. pixel intensities of an image, words from webpages, speech signal).

Target variables (response) (e.g. class labels, categories, phonemes).

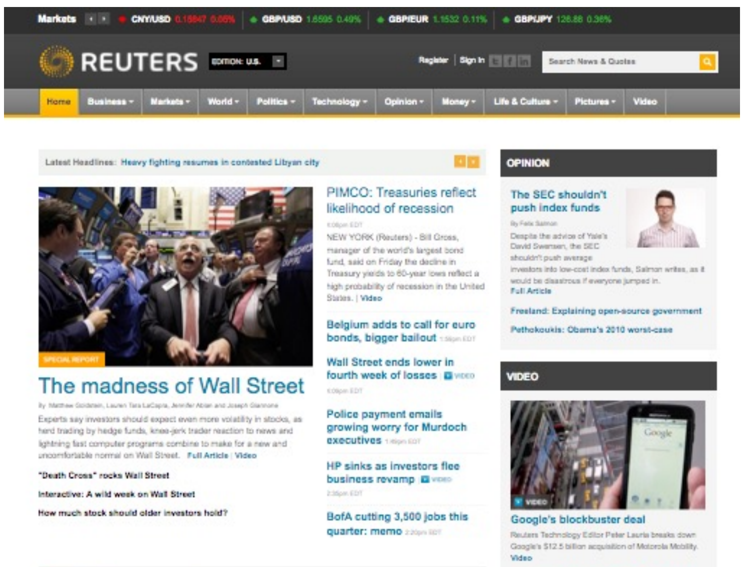
Markov Random Fields, Undirected Graphical Models.

Finding Structure in Data

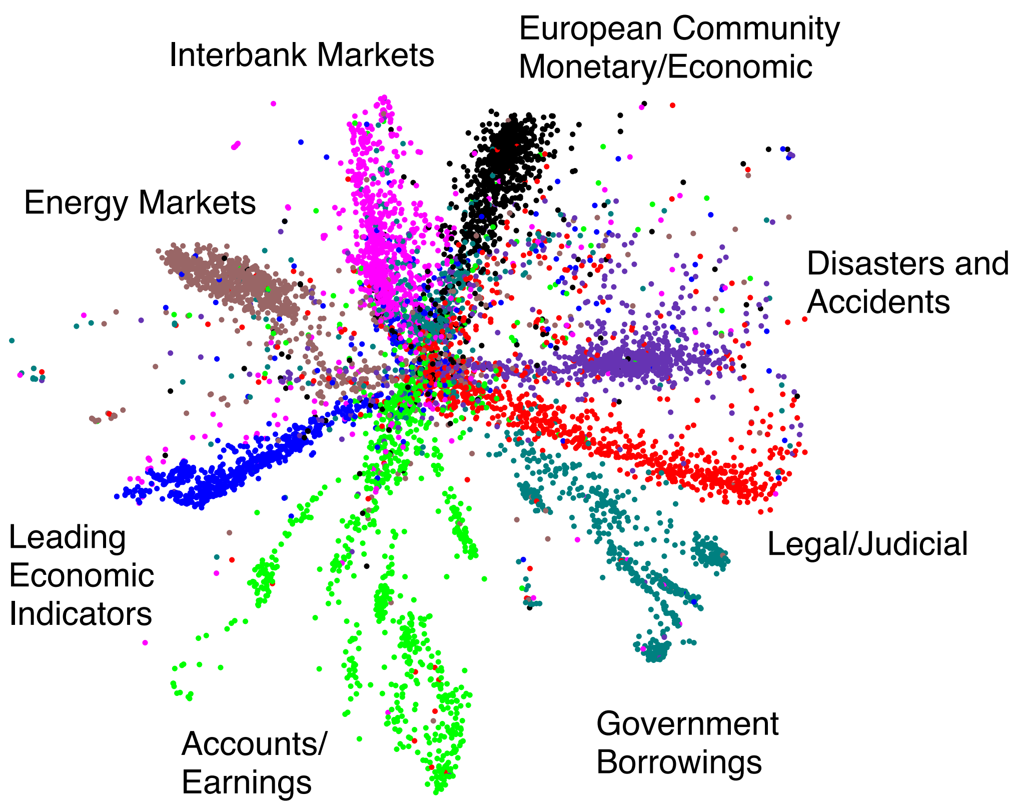
$$P(\mathbf{x}) = \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \exp\left[\mathbf{x}^\top \mathbf{W} \mathbf{h}\right]$$

Vector of word counts on a webpage

Latent variables: hidden topics



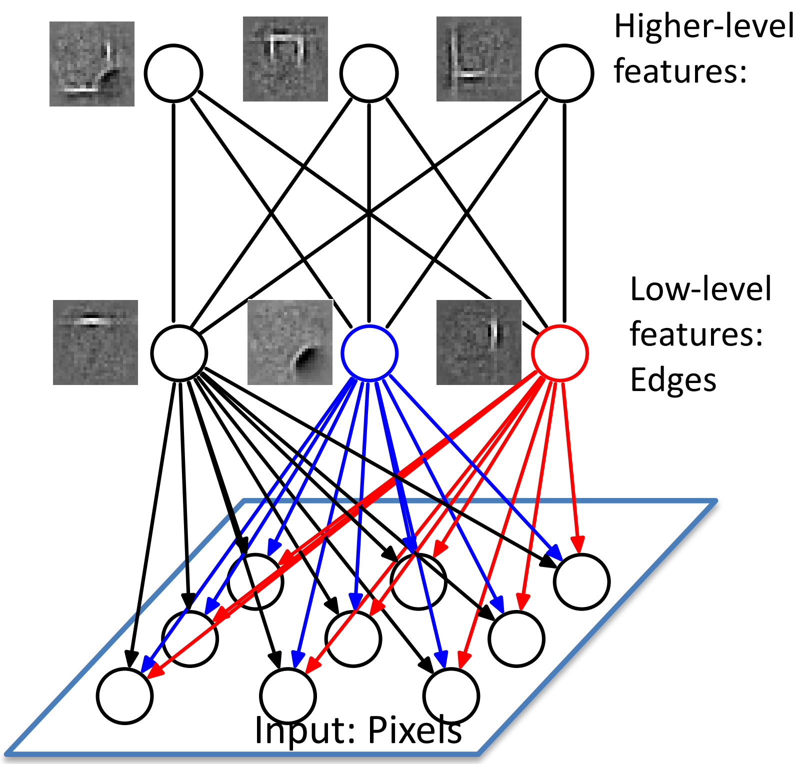
804,414 newswire stories



Important Breakthroughs

- Deep Belief Networks, 2006 (Unsupervised)

  Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets, Neural Computation, 2006.



Theoretical Breakthrough:

- Adding additional layers improves variational lower-bound.

Efficient Learning and Inference with multiple layers:

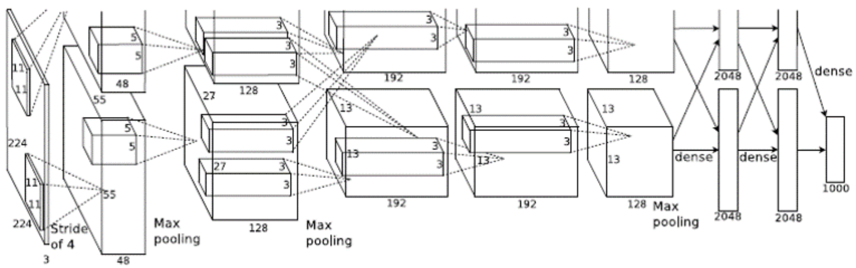
- Efficient greedy layer-by-layer learning learning algorithm.

- Inferring the states of the hidden variables in the top most layer is easy.

Important Breakthroughs

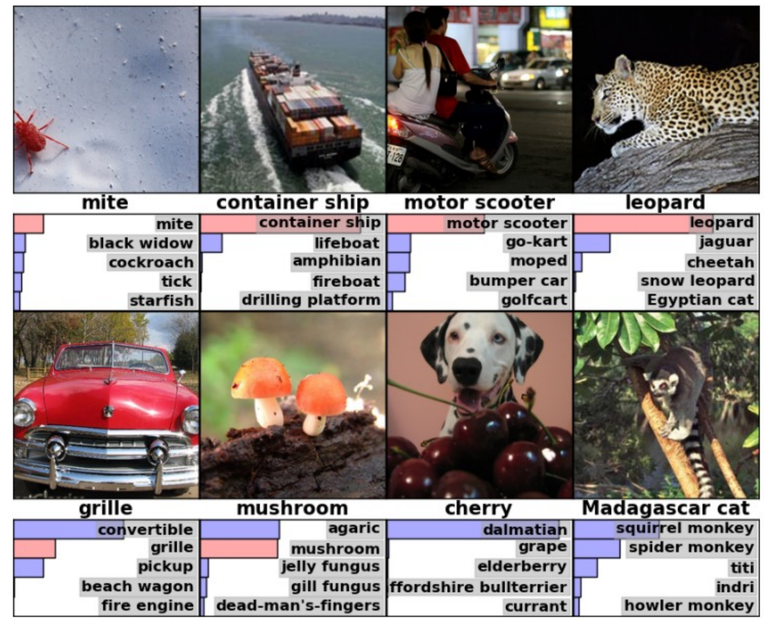
- Deep Convolutional Nets for Vision (Supervised)

  Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images
1000 classes



- Deep Nets for Speech (Supervised)

  Hinton et. al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine. 2012.

Statistical Generative Models

Training Data(CelebA)

Model Samples (Karras et.al., 2018)

4 years of progression on Faces

Brundage et al., 2017

Statistical Generative Models

- Conditional generative model P(zebra images| horse images)

- Style Transfer

Input Image

Monet

Van Gogh

Zhou el al., Cycle GAN 2017

Statistical Generative Models

- Conditional generative model P(zebra images| horse images)

▸ Failure Case

Zhou el al., Cycle GAN 2017

Course Organization

- Introduction / Background:
  - Linear Algebra, Distributions, Rules of probability.
  - Regression, Classification.
  - Feedforward neural nets, backpropagation algorithm.
  - Introduction to popular optimization and regularization techniques for deep nets.
  - Convolutional models with applications to computer vision.

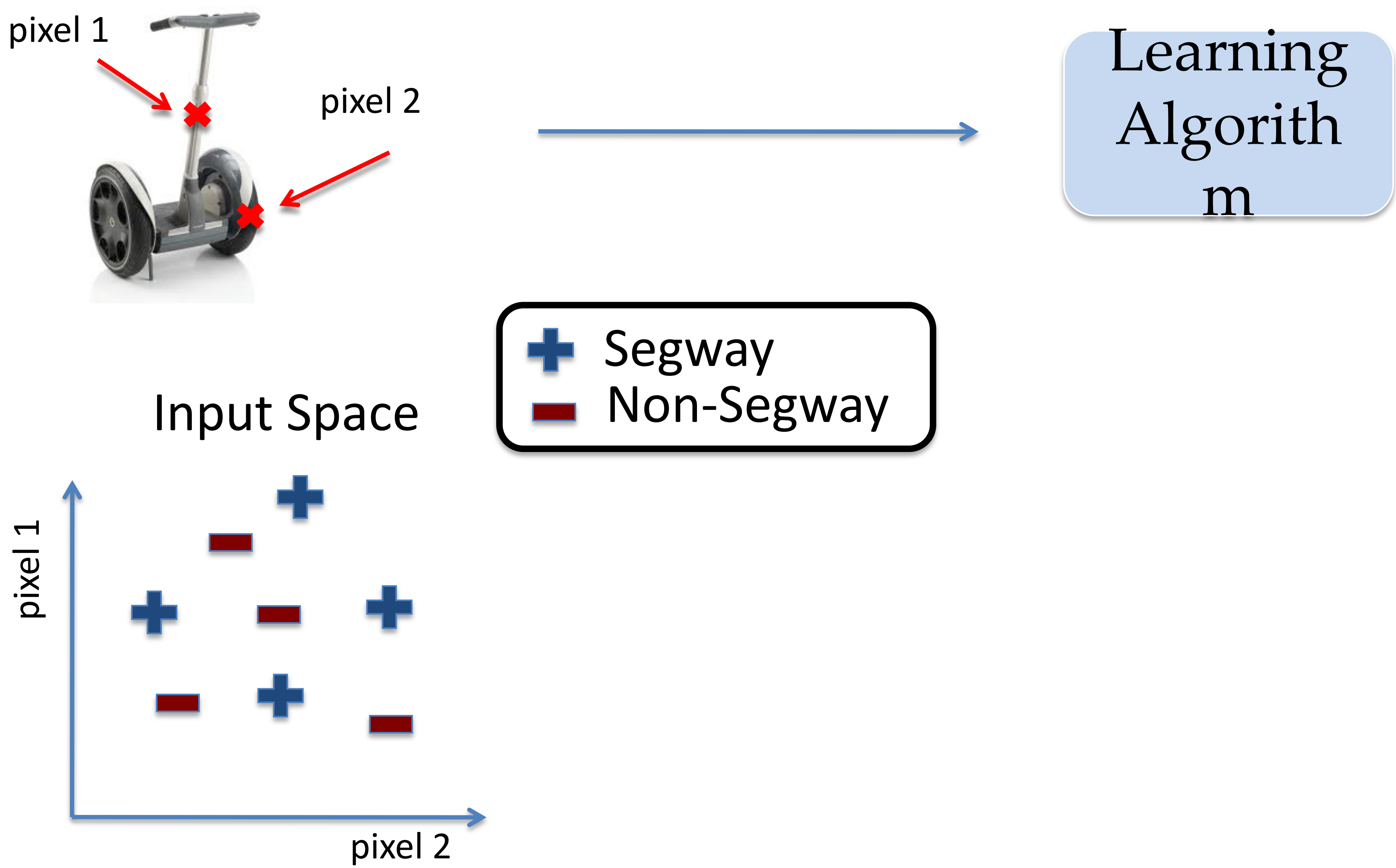
Course Organization

- Deep Learning Essentials:
  - Graphical Models: Directed and Undirected.
  - Linear Factor Models, PPCA, FA, ICA, Sparse Coding and its extensions.
  - Autoencoders and its extensions
  - Energy-based models, RBMs.
  - Monte Carlo Methods.
  - Learning and Inference: Contrastive Divergence (CD), Stochastic Maximum Likelihood Estimation, Score Matching, Ratio Matching, Pseud-likelihood Estimation.
  - Sequence Modeling: Recurrent Neural Networks, Transformers
  - Deep Generative Models: Diffusion Models, Deep Belief Networks, Deep Boltzmann Machines, Helmholtz Machines, Variational Autoencoders, Importance-weighted Autoencoders.
  - Generative Adversarial Networks (GANs), Generative Moment Matching Nets, Neural Autoregressive Density Estimator (NADE).

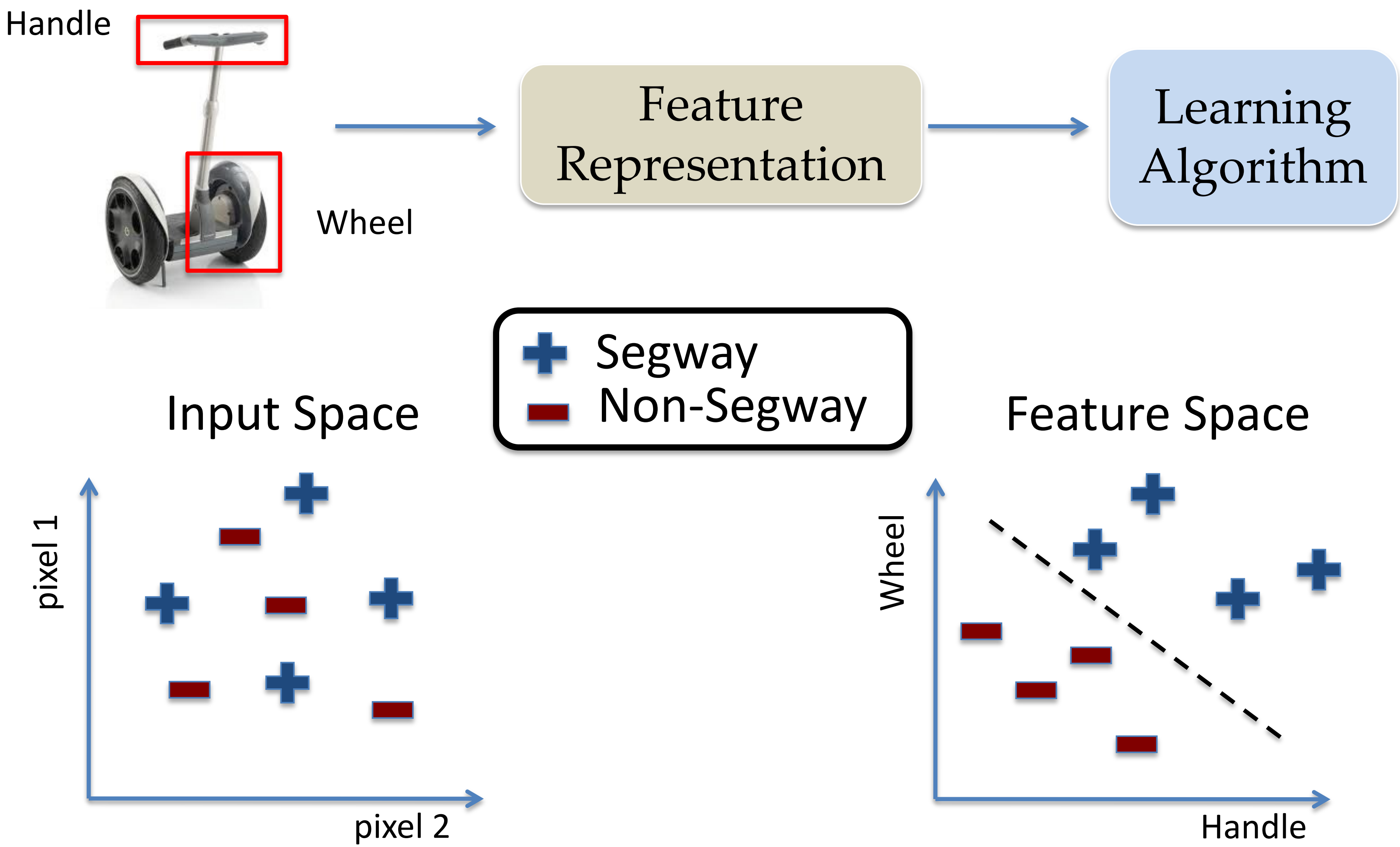
Course Organization

- Additional Topics
  - More on Regularization and Optimization in Deep Nets.
  - Sequence-to-Sequence Architectures, Attention models.
  - Some more recent topics in Deep Learning.

Learning Feature Representations



Learning Feature Representations

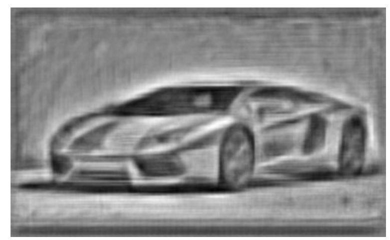


Traditional Approaches

Data → Feature extraction → Learning algorithm

Object detection

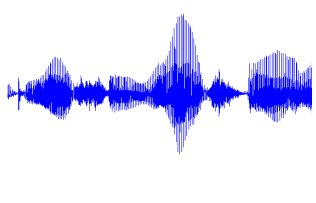
Image



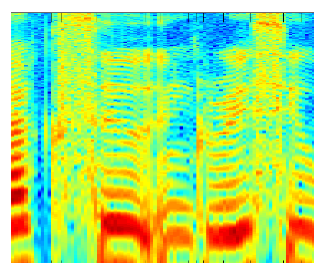
vision features

Recognition

Audio classification



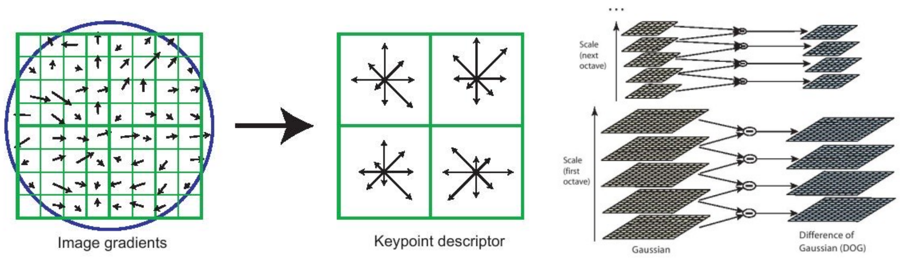
Audio



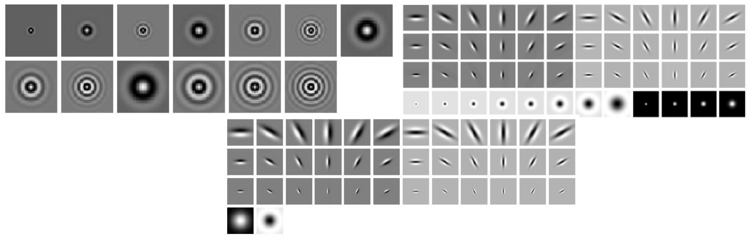
audio features

Speaker identification

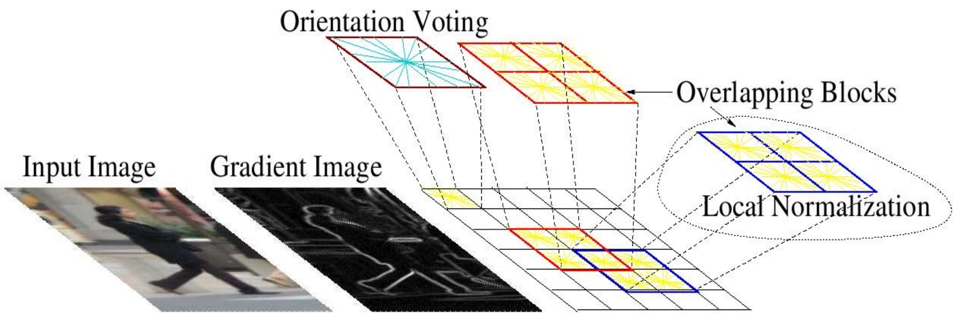
Computer Vision Features



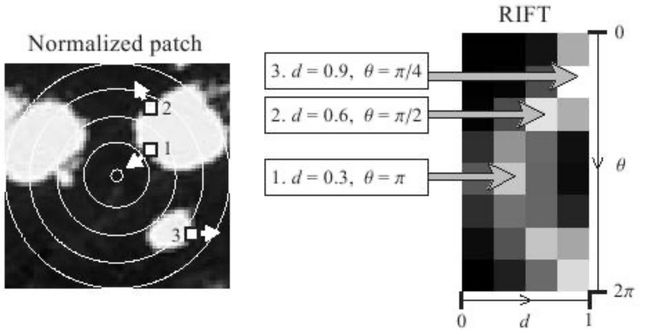
SIFT



Textons

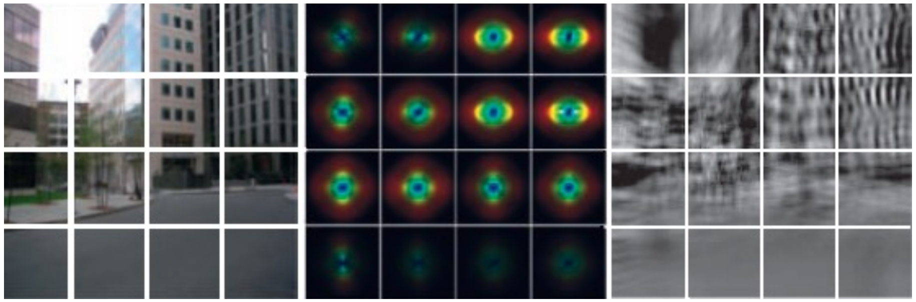


HoG

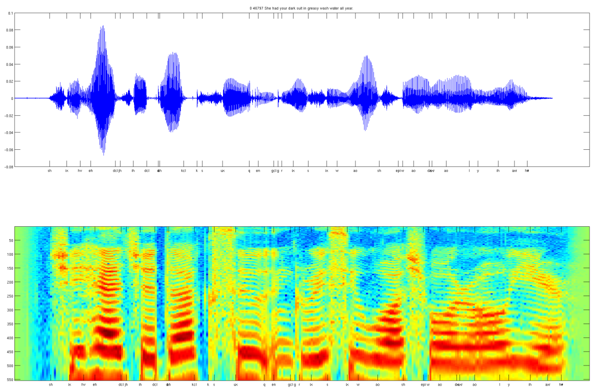


RIFT

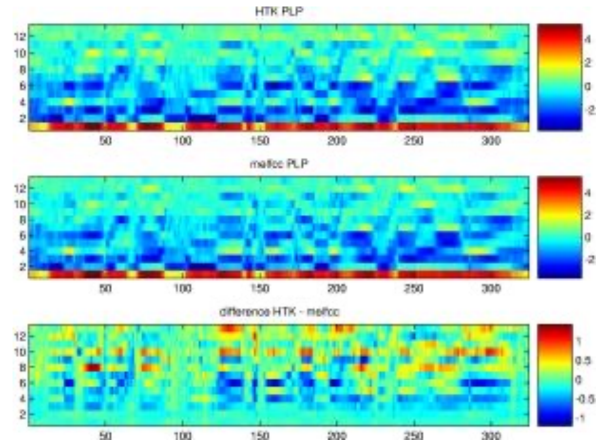
GIST



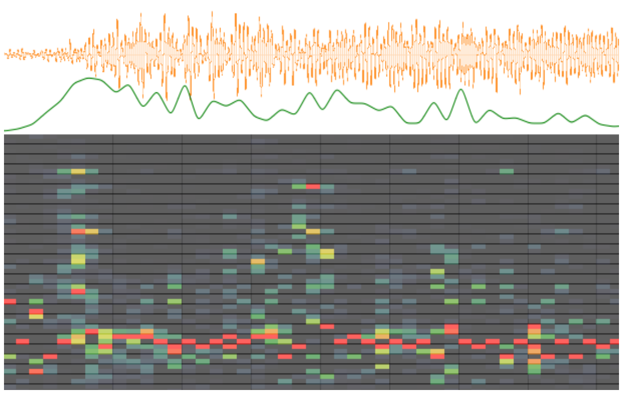
Audio Features



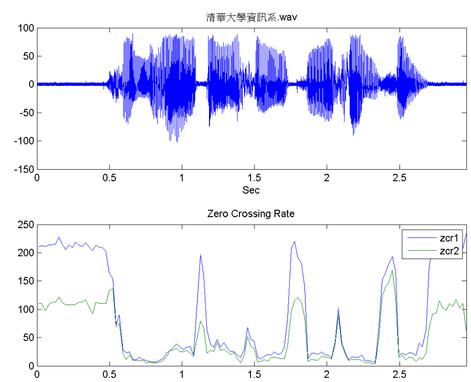
Spectrogram



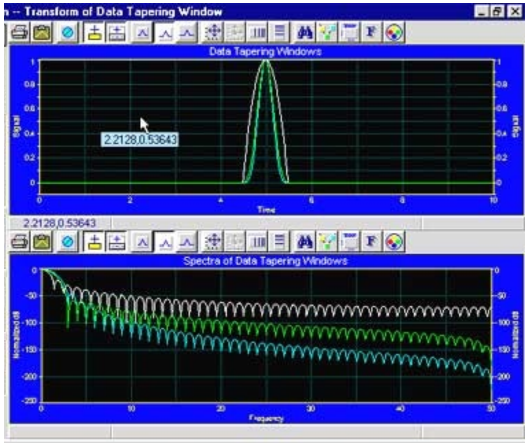
MFCC



Flux

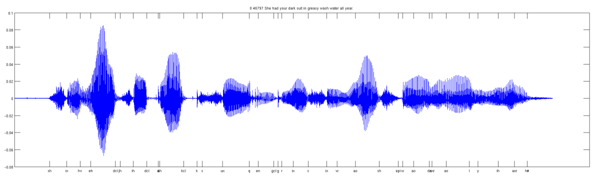


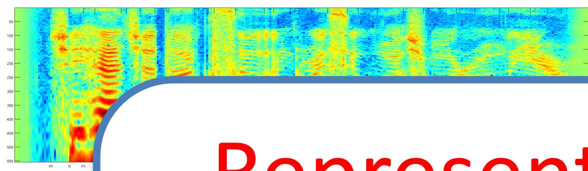
ZCR

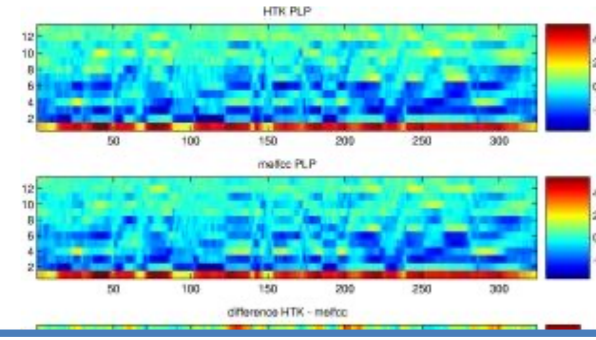


Rolloff

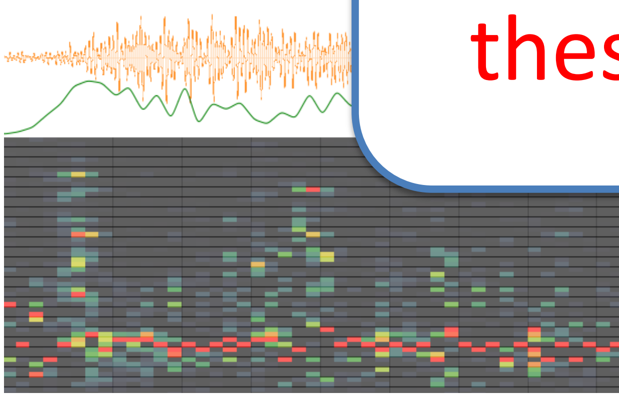
Audio Features



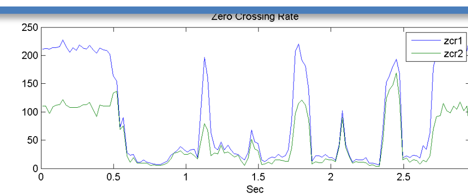




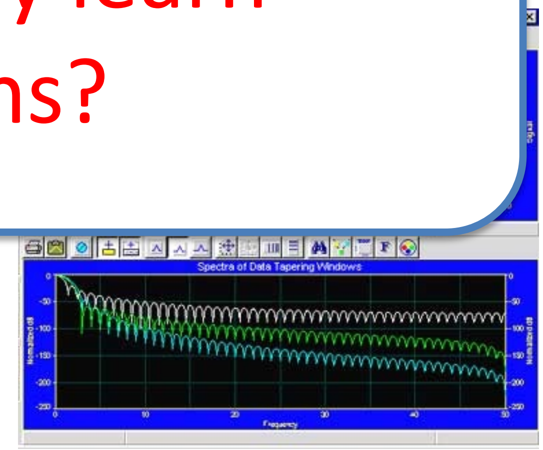
Representation Learning:
Can we automatically learn
these representations?



Flux



ZCR



Rolloff

Types of Learning

Consider observing a series of input vectors:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots$$

- **Supervised Learning:** We are also given target outputs (labels, responses): $y_1, y_2, \ldots$, and the goal is to predict correct output given a new input.

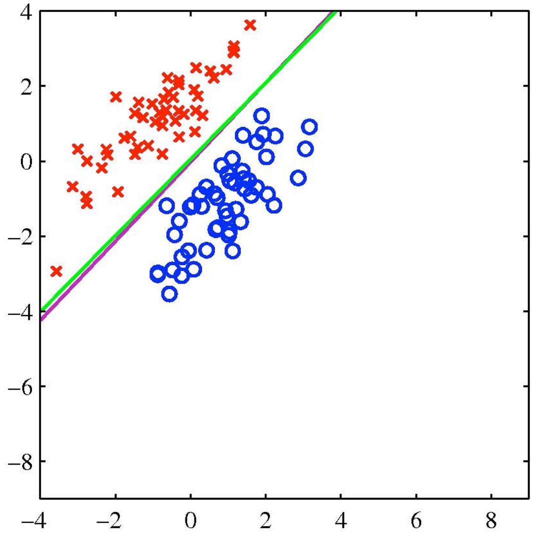
- **Unsupervised Learning:** The goal is to build a statistical model of **x**, which can be used for making predictions, decisions.

- **Reinforcement Learning:** the model (agent) produces a set of actions: $a_1, a_2, \ldots$ that affect the state of the world, and received rewards $r_1, r_2 \ldots$ The goal is to learn actions that maximize the reward.

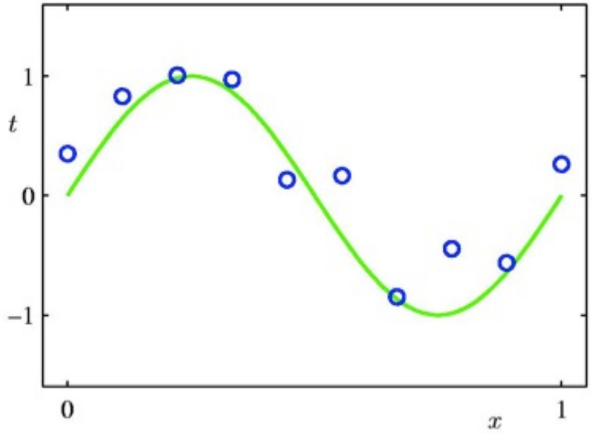
- **Semi-supervised Learning:** We are given only a limited amount of labels, but lots of unlabeled data.

Supervised Learning

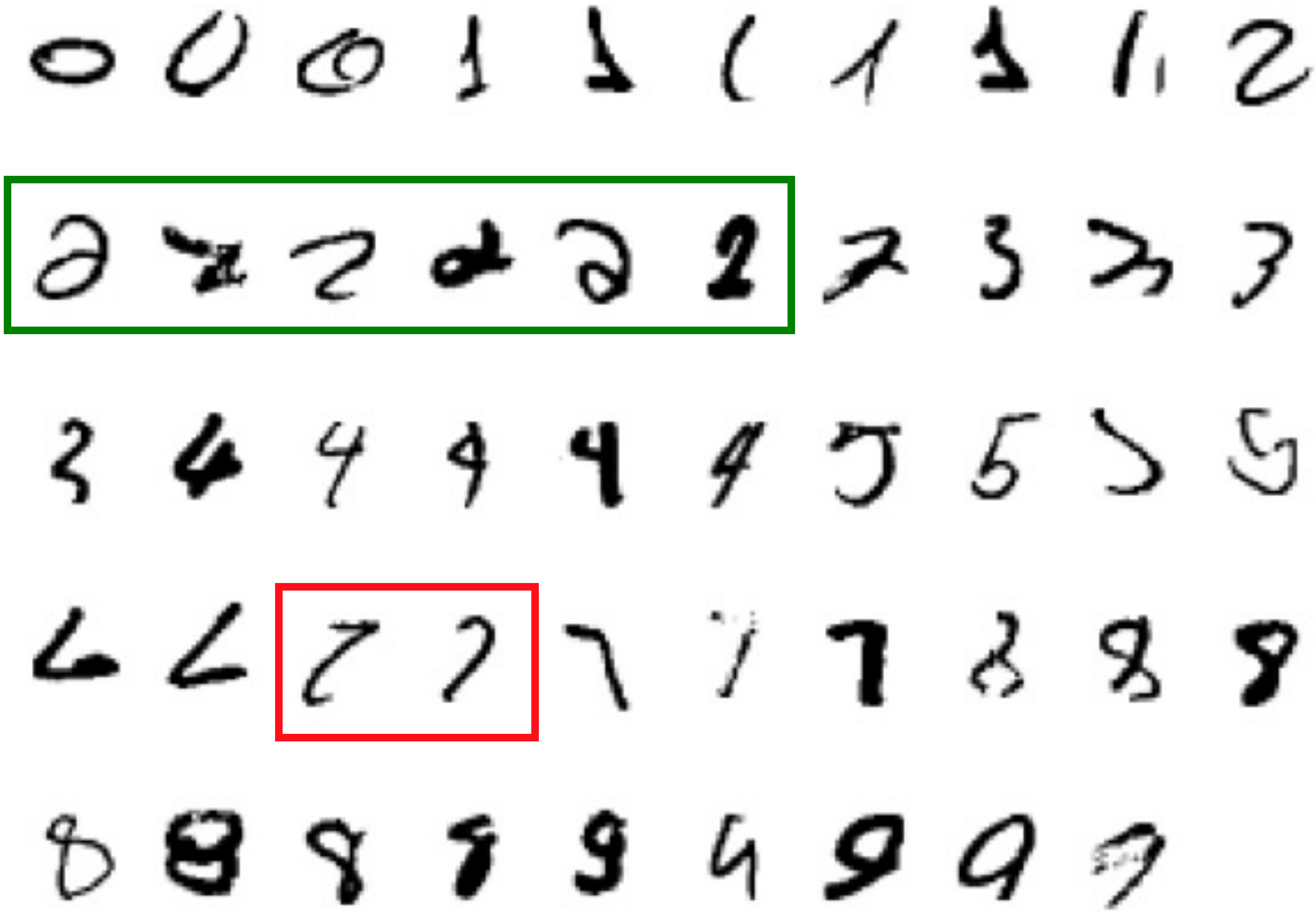
Classification: target outputs y_i are discrete class labels. The goal is to correctly classify new inputs.



Regression: target outputs y_i are continuous. The goal is to predict the output given new inputs.



Handwritten Digit Classification



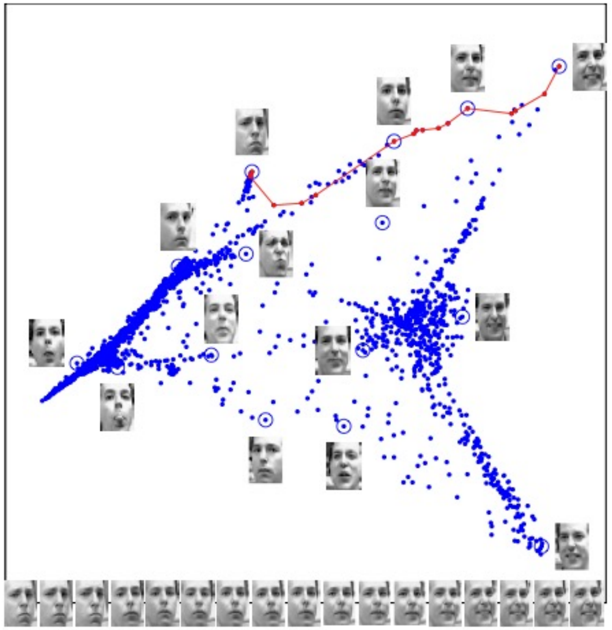
Unsupervised Learning

The goal is to construct statistical model that finds useful representation of data:

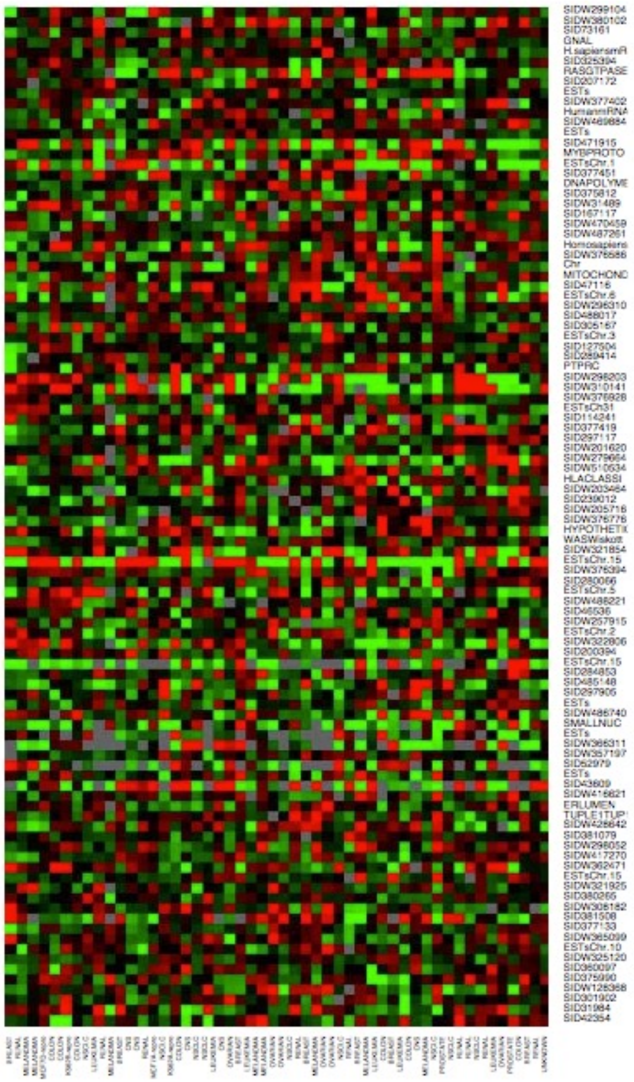
- Clustering
- Dimensionality reduction
- Modeling the data density
- Finding hidden causes (useful explanation) of the data

Unsupervised Learning can be used for:

- Structure discovery
- Anomaly detection / Outlier detection
- Data compression, Data visualization
- Used to aid classification/regression tasks



DNA Microarray Data



Expression matrix of 6830 genes (rows) and 64 samples (columns) for the human tumor data.

The display is a heat map ranging from bright green (under expressed) to bright red (over expressed).

Questions we may ask:

• Which samples are similar to other samples in terms of their expression levels across genes.

• Which genes are similar to each other in terms of their expression levels across samples.

Linear Least Squares

- Given a vector of d-dimensional inputs $\mathbf{x} = (x_1, x_2, ..., x_d)^T$, we want to predict the target (response) using the linear model:

$$y(x, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j.$$

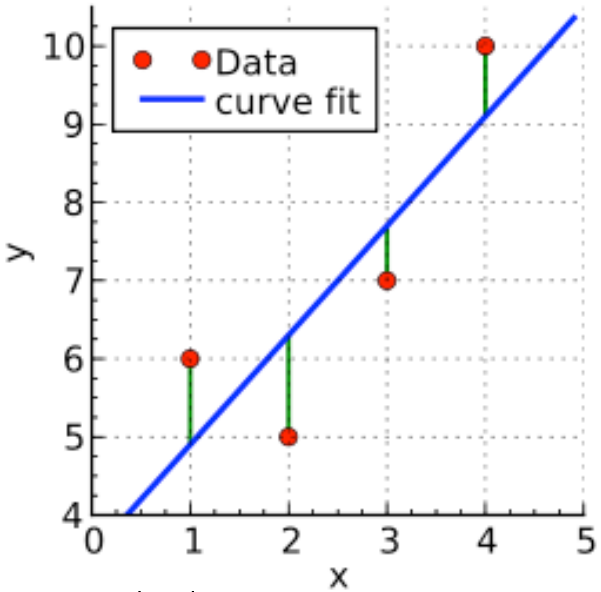
- The term w_0 is the intercept, or often called bias term. It will be convenient to include the constant variable 1 in **x** and write:

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T \mathbf{w}.$$

- Observe a training set consisting of N observations $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)^T$, together with the corresponding target values $\mathbf{t} = (t_1, t_2, ..., t_N)^T$.
- Note that **X** is an $N \times (d + 1)$ matrix.

Linear Least Squares

One option is to minimize **the sum of the squares of the errors** between the predictions $y(\mathbf{x}_n, \mathbf{w})$ for each data point x_n and the corresponding real-valued targets t_n.



Source: Wikipedia

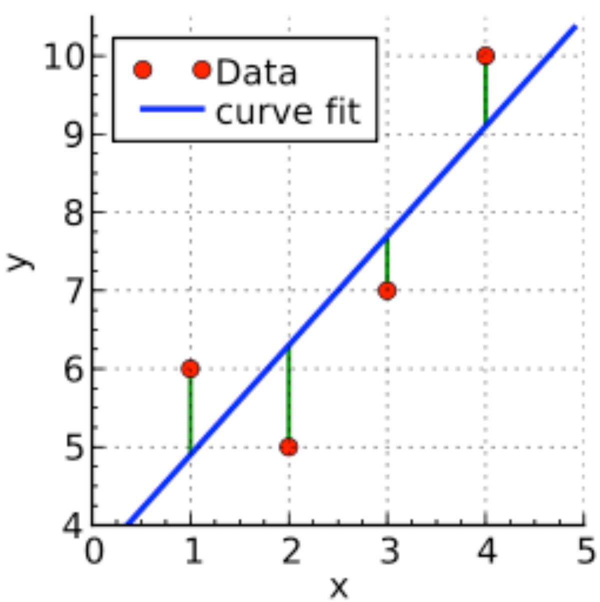
Loss function: sum-of-squared error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n^T\mathbf{w} - t_n)^2$$

$$= \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{t})^{\mathbf{T}}(\mathbf{X}\mathbf{w} - \mathbf{t}).$$

Linear Least Squares

If $\mathbf{X}^{\mathbf{T}}\mathbf{X}$ is nonsingular, then the unique solution is given by:



Source: Wikipedia

optimal weights

vector of target values

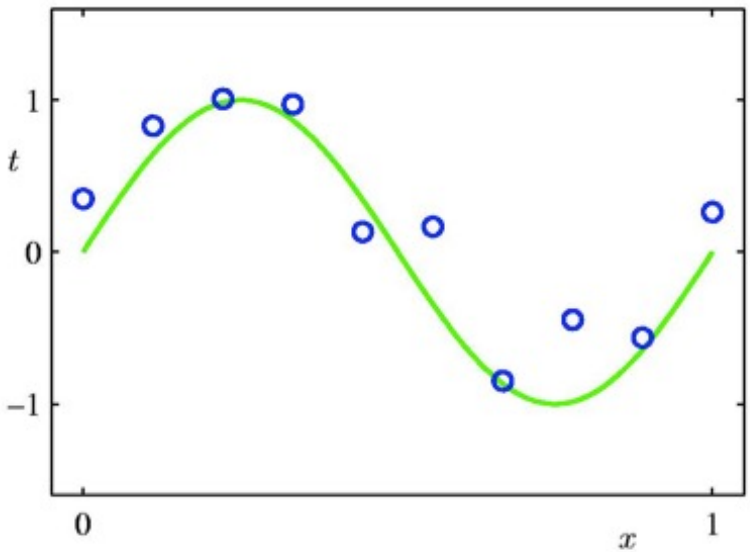
$$\mathbf{w}^* = (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{t}$$

the design matrix has one input vector per row

- At an arbitrary input $\mathbf{x}_0$, the prediction is $y(\mathbf{x}_0, \mathbf{w}) = \mathbf{x}_0^T\mathbf{w}^*$.
- The entire model is characterized by d+1 parameters **w***.

Example: Polynomial Curve Fitting

Consider observing a training set consisting of N 1-dimensional observations:
$\mathbf{x} = (x_1, x_2, ..., x_N)^T$, together with corresponding real-valued targets:
$\mathbf{t} = (t_1, t_2, ..., t_N)^T$.



- The green plot is the true function $\sin(2\pi x)$.
- The training data was generated by taking x_n spaced uniformly between [0 1].
- The target set (blue circles) was obtained by first computing the corresponding values of the sin function, and then adding a small Gaussian noise.

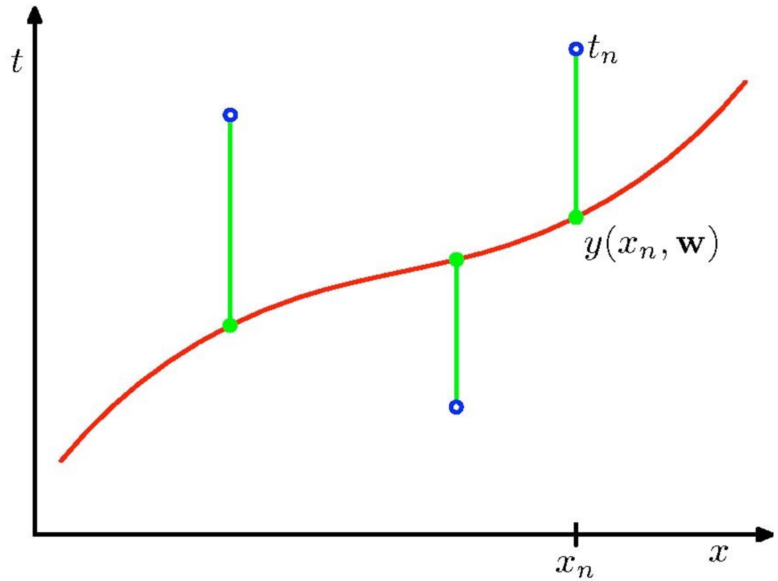
Goal: Fit the data using a polynomial function of the form:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

Note: the polynomial function is a nonlinear function of x, but it is a linear function of the coefficients **w** $\rightarrow$ **Linear Models**.

Example: Polynomial Curve Fitting

- As for the least squares example: we can minimize the sum of the squares of the errors between the predictions $y(x_n, \mathbf{w})$ for each data point x_n and the corresponding target values t_n.

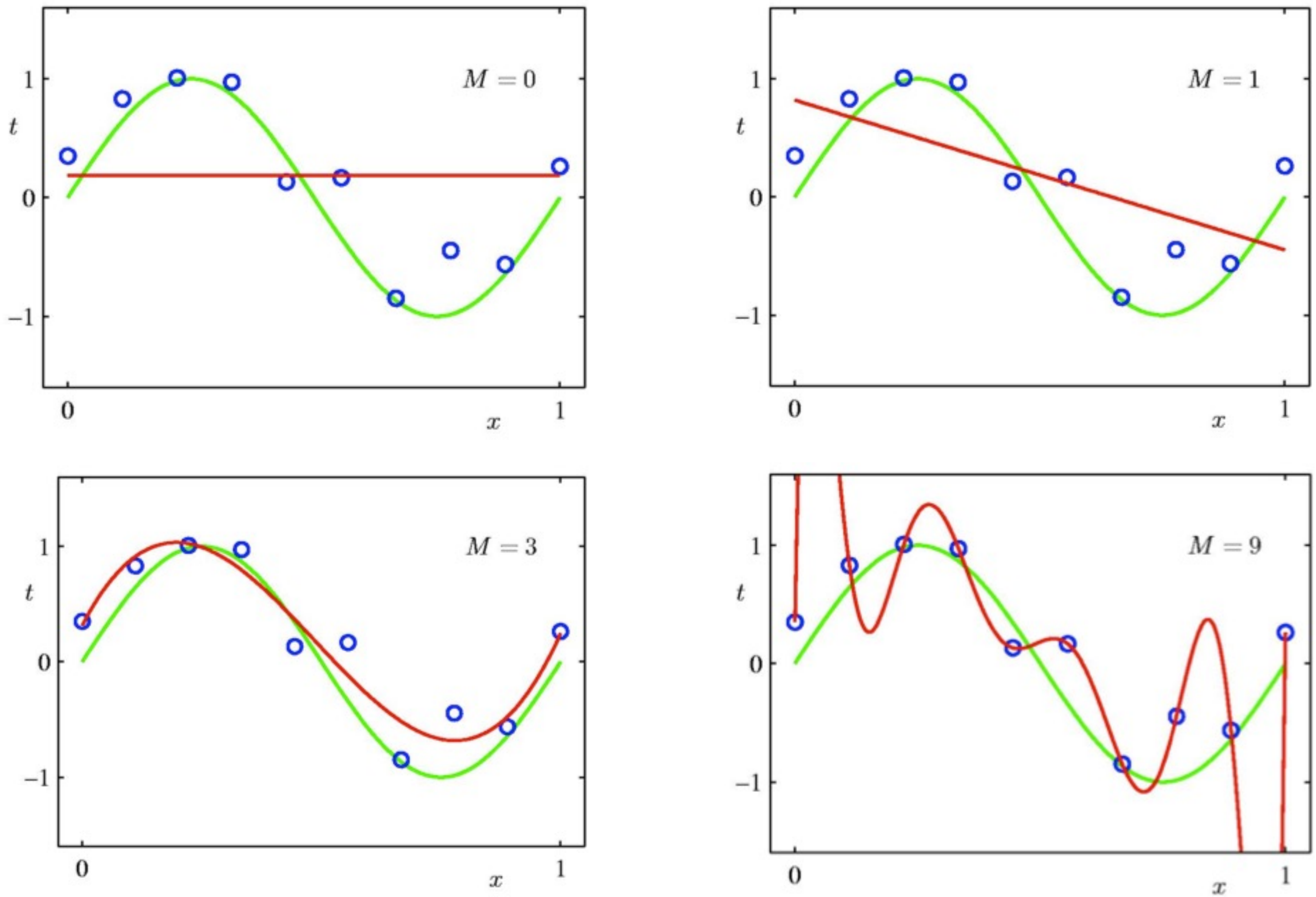


Loss function: sum-of-squared error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2.$$

- Similar to the linear least squares: Minimizing sum-of-squared error function has a unique solution $\mathbf{w}^*$.
- The model is characterized by M+1 parameters $\mathbf{w}^*$.
- How do we choose M? $\rightarrow$ **Model Selection**.

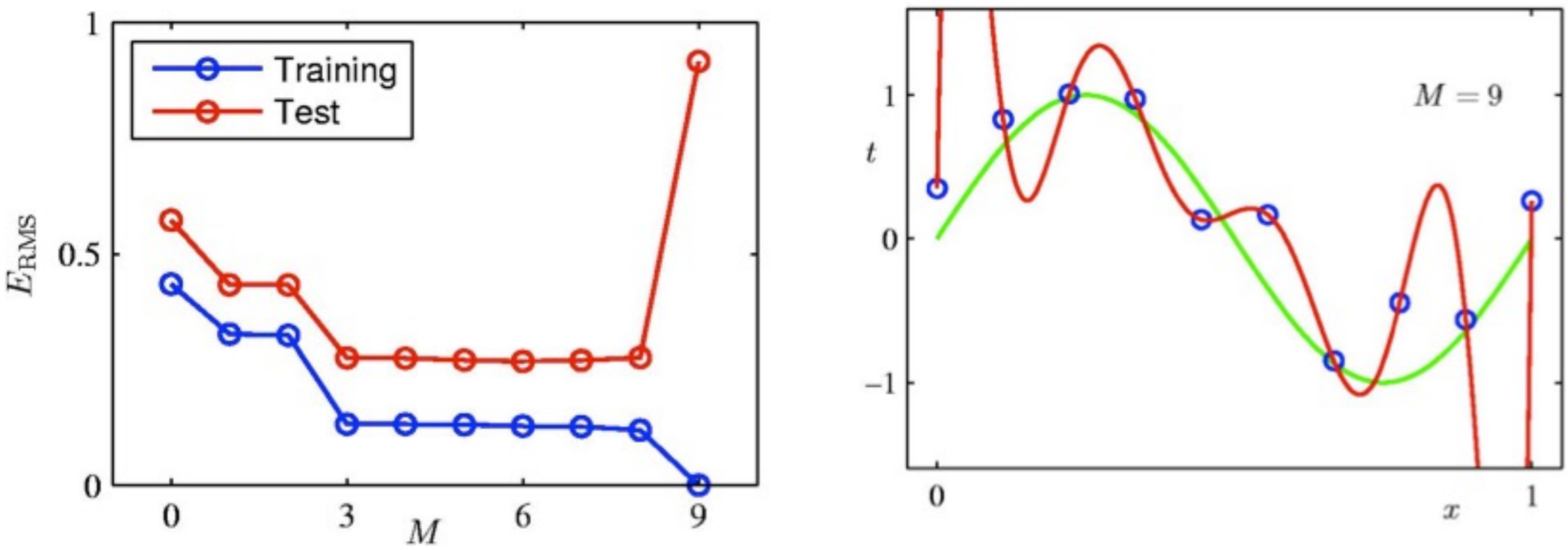
Some Fits to the Data



For M=9, we have fitted the training data perfectly.

Overfitting

• Consider a separate **test set** containing 100 new data points generated using the same procedure that was used to generate the training data.

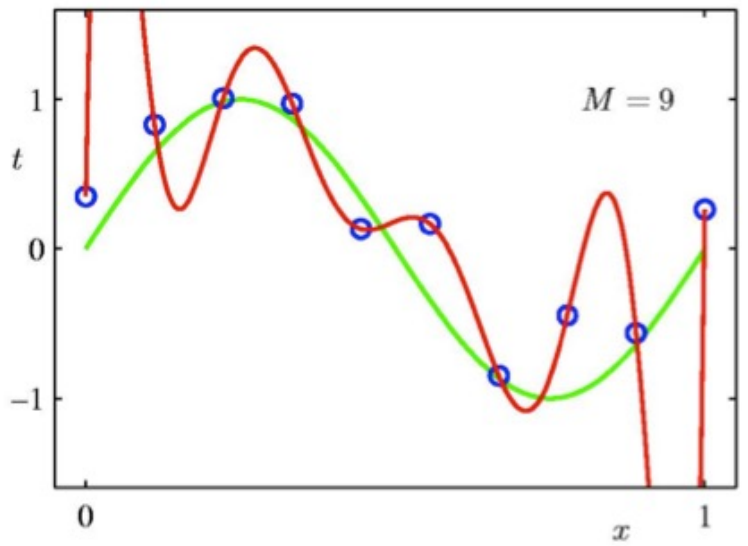


• For M=9, the training error is zero $\rightarrow$ The polynomial contains 10 degrees of freedom corresponding to 10 parameters **w**, and so can be fitted exactly to the 10 data points.

• However, the test error has become very large. Why?

Overfitting

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

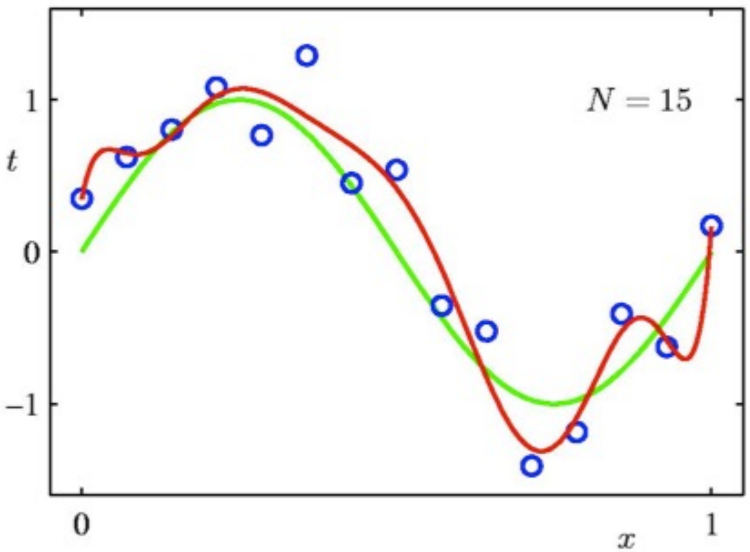


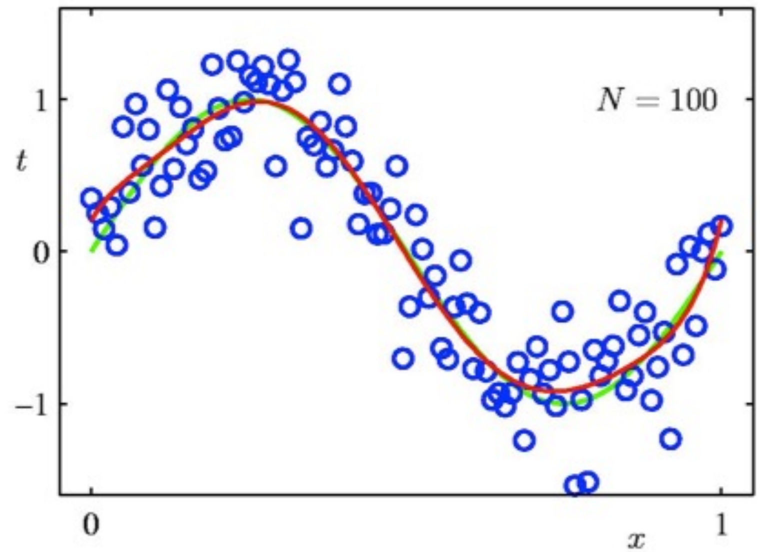
- As M increases, the magnitude of coefficients gets larger.
- For M=9, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

More flexible polynomials with larger M tune to the random noise on the target values.

Varying the Size of the Data

9th order polynomial



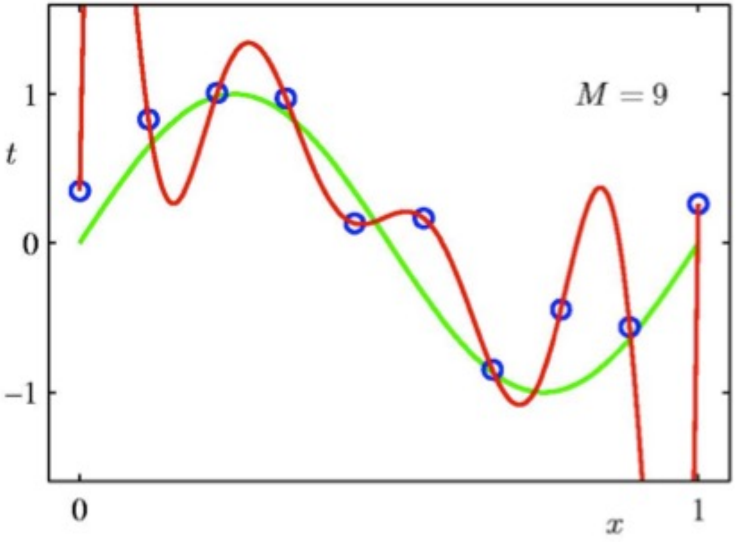


• For a given model complexity, the overfitting problem becomes less severe as the size of the dataset increases.

• However, the number of parameters is not necessarily the most appropriate measure of the model complexity.

Generalization

- The goal is achieve good **generalization** by making accurate predictions for new test data that is not known during learning.
- Choosing the values of parameters that minimize the loss function on the training data may not be the best option.
- We would like to model the true regularities in the data and ignore the noise in the data:
  - It is hard to know which regularities are real and which are accidental due to the particular training examples we happen to pick.



- Intuition: We expect the model to generalize if it explains the data well given the complexity of the model.
- If the model has as many degrees of freedom as the data, it can fit the data perfectly. But this is not very informative.
- Some theory on how to control model complexity to optimize generalization.

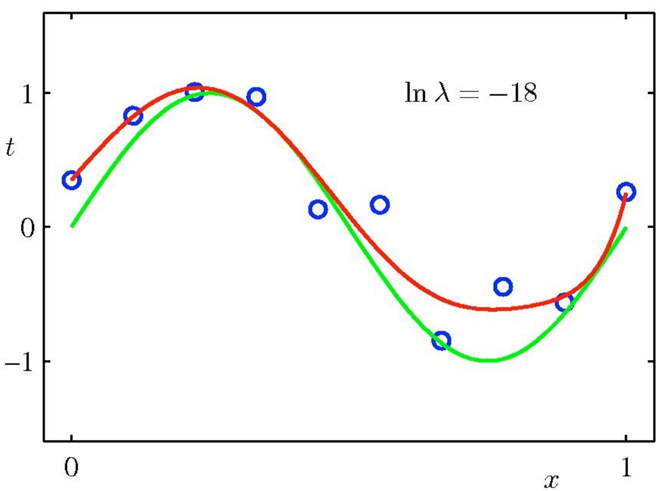
A Simple Way to Penalize Complexity

One technique for controlling over-fitting phenomenon is **regularization**, which amounts to adding a penalty term to the error function.

penalized error function → target value → regularization parameter

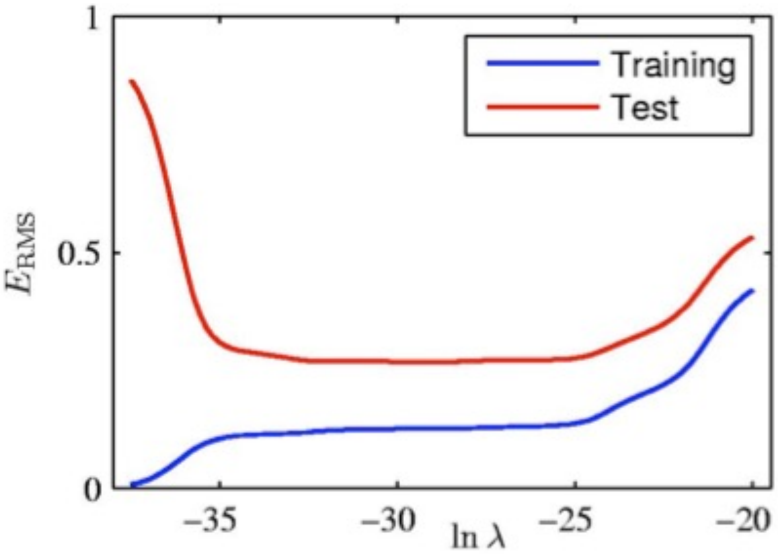
$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where $||\mathbf{w}|| = \mathbf{w}^T\mathbf{w} = w_1^2 + w_2^2 + ... + w_M^2$ called the regularization term. Note that we do not penalize the bias term w_0.



- The idea is to "shrink" estimated parameters towards zero (or towards the mean of some other weights).
- Shrinking to zero: penalize coefficients based on their size.
- For a penalty function which is the sum of the squares of the parameters, this is known as "**weight decay**", or "**ridge regression**".

Regularization



	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Graph of the root-mean-squared training and test errors vs. lnλ for the M=9 polynomial.

How to choose λ?

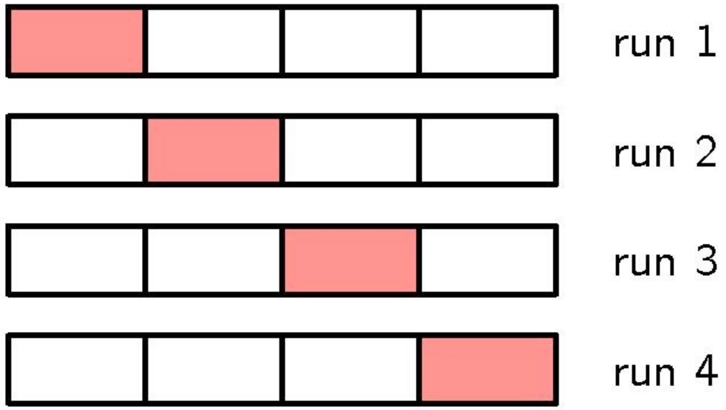
Cross Validation

If the data is plentiful, we can divide the dataset into three subsets:

- Training Data: used to fitting/learning the parameters of the model.
- Validation Data: not used for learning but for selecting the model, or choosing the amount of regularization that works best.
- Test Data: used to get performance of the final model.

For many applications, the supply of data for training and testing is limited.
To build good models, we may want to use as much training data as possible.
If the validation set is small, we get noisy estimate of the predictive performance.

S fold cross-validation



- The data is partitioned into S groups.
- Then S-1 of the groups are used for training the model, which is evaluated on the remaining group.
- Repeat procedure for all S possible choices of the held-out group.
- Performance from the S runs are averaged.

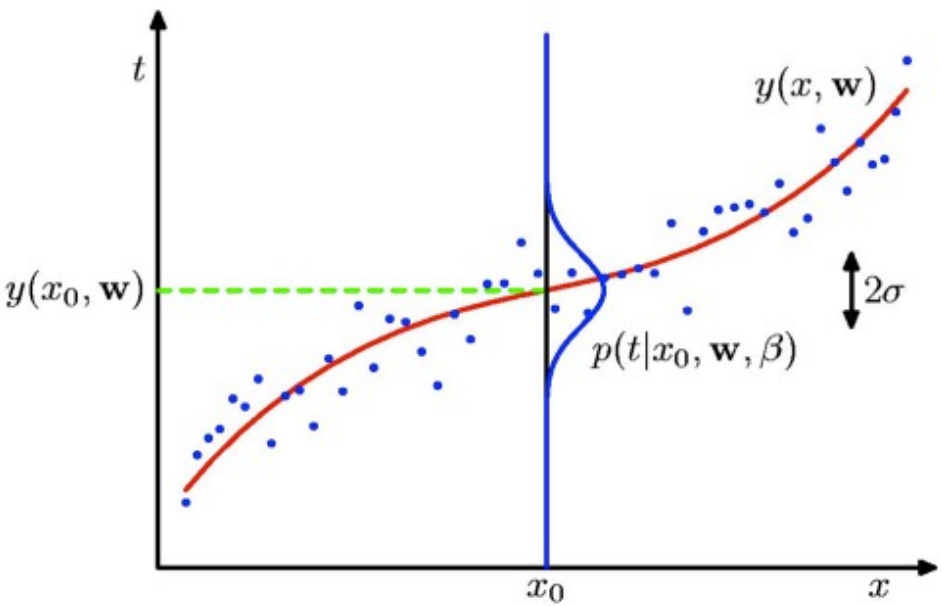
Probabilistic Perspective

- So far we saw that polynomial curve fitting can be expressed in terms of error minimization. We now view it from probabilistic perspective.

- Suppose that our model arose from a statistical model:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

where ϵ is a random error having Gaussian distribution with zero mean, and is independent of **x**.



Thus we have:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}),$$

where β is a precision parameter, corresponding to the inverse variance.

I will use probability distribution and probability density interchangeably. It should be obvious from the context.

Sampling Assumption

- Assume that the training examples are drawn independently from the set of all possible examples, or from the same underlying distribution $p(\mathbf{x}, t)$.
- We also assume that the training examples are identically distributed $\rightarrow$ i.i.d assumption.
- Assume that the test samples are drawn in exactly the same way -- i.i.d from the same distribution as the training data.
- These assumptions make it unlikely that some strong regularity in the training data will be absent in the test data.

Maximum Likelihood

If the data are assumed to be independently and identically distributed (*i.i.d assumption*), the likelihood function takes form:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}).$$

It is often convenient to maximize the log of the likelihood function:

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\underbrace{\frac{\beta}{2} \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2}_{\beta E(\mathbf{w})} + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

- Maximizing log-likelihood with respect to **w** (under the assumption of a Gaussian noise) is equivalent to minimizing the *sum-of-squared error* function.

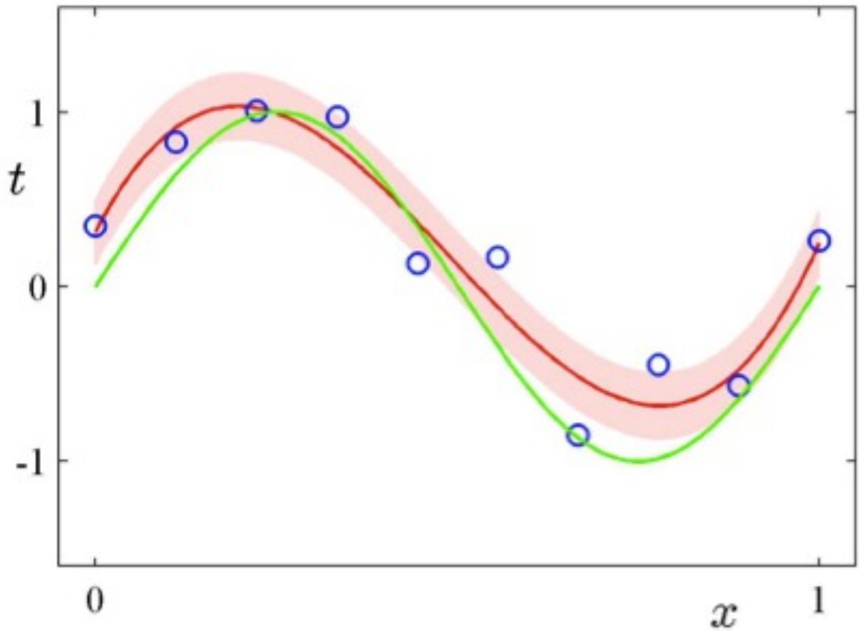
- Determine $\mathbf{w}_{ML}$ by maximizing log-likelihood. Then maximizing w.r.t. β:

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n} (y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n)^2.$$

Predictive Distribution

Once we determined the parameters **w** and β, we can make prediction for new values of **x**:

$$p(t|\mathbf{x}, \mathbf{w}_{ML}, \beta_{ML}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{ML}), \beta_{ML}^{-1}).$$



Statistical Decision Theory

- We now develop a small amount of theory that provides a framework for developing many of the models we consider.

- Suppose we have a real-valued input vector **x** and a corresponding target (output) value t with joint probability distribution: $p(\mathbf{x}, t)$.

- Our goal is predict target t given a new value for **x**:
  - for regression: t is a real-valued continuous target.
  - for classification: t a categorical variable representing class labels.

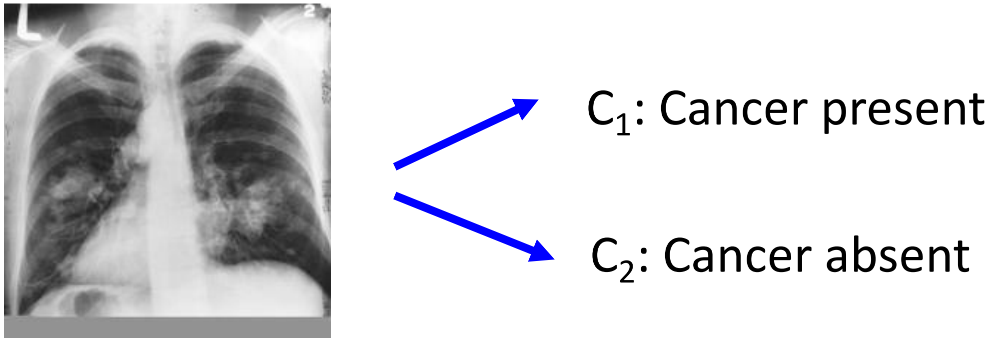
The joint probability distribution $p(\mathbf{x}, t)$ provides a complete summary of uncertainties associated with these random variables.

Determining $p(\mathbf{x}, t)$ from training data is known as the inference problem.

Example: Classification

Medical diagnosis: Based on the X-ray image, we would like determine whether the patient has cancer or not.

- The input vector **x** is the set of pixel intensities, and the output variable t will represent the presence of cancer, class C_1, or absence of cancer, class C_2.



x -- set of pixel intensities

- Choose t to be binary: t=0 correspond to class C_1, and t=1 corresponds to C_2.

Inference Problem: Determine the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$ or equivalently $p(\mathbf{x}, t)$. However, in the end, we must make a decision of whether to give treatment to the patient or not.

Example: Classification

Informally: Given a new X-ray image, our goal is to decide which of the two classes that image should be assigned to.

- We could compute conditional probabilities of the two classes, given the input image:

posterior probability of C_k given observed data.

probability of observed data given C_k

prior probability for class C_k

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}, \mathcal{C}_k)}{\sum_{k=1}^{K} p(\mathbf{x}, \mathcal{C}_k)} = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

Bayes' Rule

- If our goal to minimize the probability of assigning **x** to the wrong class, then we should choose the class having the highest posterior probability.

Expected Loss

- Loss Function: overall measure of loss incurred by taking any of the available decisions.
- Suppose that for **x**, the true class is C_k, but we assign **x** to class j
  $\rightarrow$ incur loss of L_{kj} (k,j element of a loss matrix).

Consider medical diagnosis example: example of a loss matrix:

		Decision	
		cancer	normal
Truth	cancer	0	1000
	normal	1	0

Expected Loss:

$$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj}\, p(\mathbf{x}, \mathcal{C}_k)\, \mathrm{d}\mathbf{x}$$

Goal is to choose decision regions $\mathcal{R}_j$ as to minimize expected loss.

Regression

Let $\mathbf{x} \in R^d$ denote a real-valued input vector, and t $\in$ R denote a real-valued random target (output) variable with joint the distribution $p(\mathbf{x}, t)$.

- The decision step consists of finding an estimate y(**x**) of t for each input **x**.

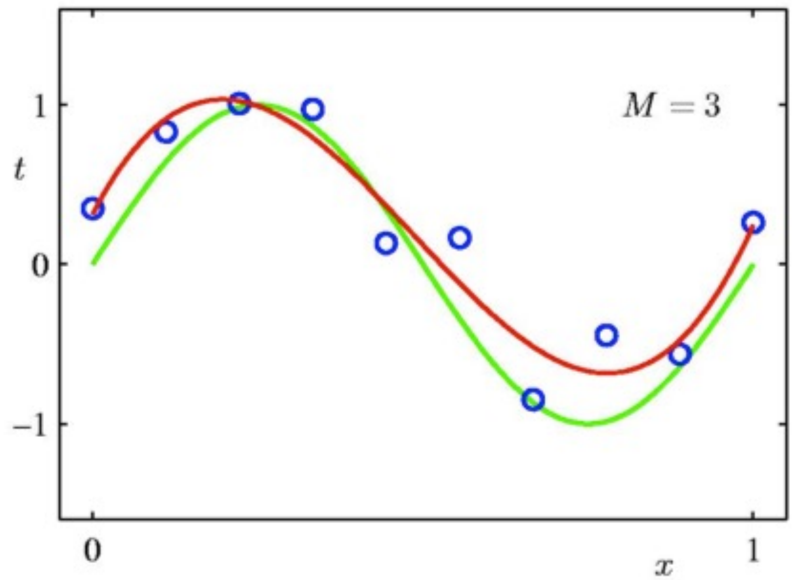
- To quantify what it means to do well or poorly on a task, we need to define a loss (error) function: $L(t, y(\mathbf{x}))$.

- The average, or expected, loss is given by:

$$\mathbb{E}[L] = \int\int L(t, y(\mathbf{x}))p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t.$$

- If we use squared loss, we obtain:

$$\mathbb{E}[L] = \int\int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t.$$



Squared Loss Function

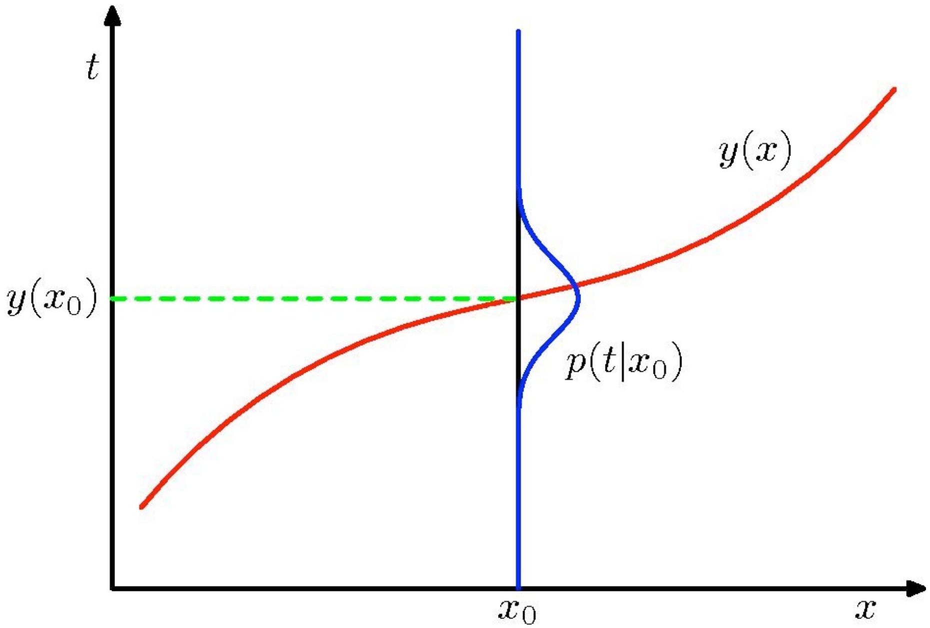
- If we use squared loss, we obtain:

$$\mathbb{E}[L] = \int\int \left(t - y(\mathbf{x})\right)^2 p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t.$$

- Our goal is to choose y(x) so as to minimize the expected squared loss.

- The optimal solution (if we assume a completely flexible function) is the conditional average:

$$y(\mathbf{x}) = \int t p(t|\mathbf{x})\mathrm{d}t = \mathbb{E}[t|\mathbf{x}].$$



The regression function y(**x**) that minimizes the expected squared loss is given by the mean of the conditional distribution $p(t|\mathbf{x})$.

Squared Loss Function

- If we use squared loss, we obtain:

$$
\begin{aligned}
(y(\mathbf{x}) - t)^2 &= (y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}] + \mathbb{E}[t|\mathbf{x}] - t)^2 \\
&= \big(y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\big)^2 + 2\big(y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\big)\big(\mathbb{E}[t|\mathbf{x}] - t\big) + \big(\mathbb{E}[t|\mathbf{x}] - t\big)^2.
\end{aligned}
$$

- Plugging into expected loss:

$$
\mathbb{E}[L] = \underbrace{\int \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 \, p(\mathbf{x}) \, \mathrm{d}\mathbf{x}}_{} + \underbrace{\int \mathrm{var}\,[t|\mathbf{x}] \, p(\mathbf{x}) \, \mathrm{d}\mathbf{x}}_{}
$$

expected loss is minimized when $y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]$.

intrinsic variability of the target values.

Because it is independent noise, it represents an irreducible minimum value of expected loss.

Other Loss Function

- Simple generalization of the squared loss, called the *Minkowski* loss:

$$\mathbb{E}[L] = \int\int \left(t - y(\mathbf{x})\right)^q p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t.$$

- The minimum of $\mathbb{E}[L]$ is given by:
  - the conditional mean for q=2,
  - the conditional median when q=1, and
  - the conditional mode for q $\rightarrow$ 0.

Discriminative vs. Generative

- Generative Approach:

Model the joint density: $p(\mathbf{x}, t) = p(\mathbf{x}|t)p(t)$,

or joint distribution: $p(\mathbf{x}, \mathcal{C}_k) = p(\mathbf{x}|\mathcal{C}_k)p(C_k)$.

Infer conditional density:

$$p(t|\mathbf{x}) = \frac{p(\mathbf{x}|t)p(t)}{p(\mathbf{x})}.$$

- Discriminative Approach:

Model conditional density $p(t|\mathbf{x})$ directly.

Linear Basis Function Models

- Remember, the simplest linear model for regression:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j,$$

where $\mathbf{x} = (x_1, x_2, ..., x_d)^T$ is a d-dimensional input vector (covariates).

Key property: linear function of the parameters $w_0, w_1, ..., w_d$.

- However, it is also a linear function of the input variables.
  Instead consider:

$$y(\mathbf{x}, \mathbf{w}) = w_0 \phi_0(\mathbf{x}) + w_1 \phi_1(\mathbf{x}) + ... + w_{M-1} \phi_{M-1}(\mathbf{x}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}),$$

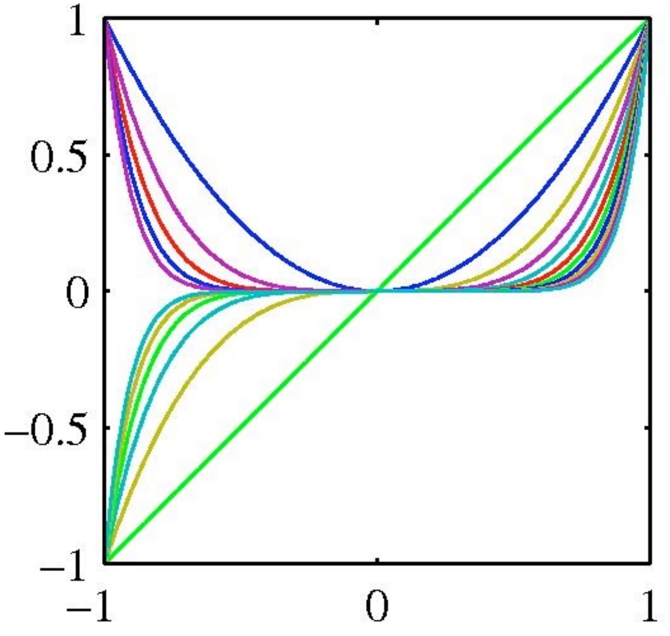
where $\phi_j(\mathbf{x})$ are known as basis functions.

- Typically $\phi_0(\mathbf{x}) = 1$ so that w_0 acts as a bias (or intercept).
- In the simplest case, we use linear bases functions: $\phi_j(\mathbf{x}) = x_j$.
- Using nonlinear basis allows the functions $y(\mathbf{x}, \mathbf{w})$ to be nonlinear functions of the input space.

Linear Basis Function Models

Polynomial basis functions:

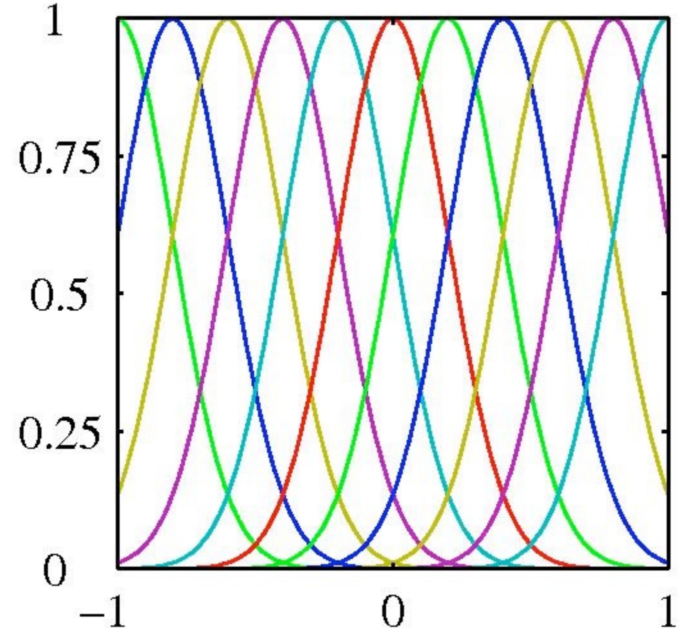
$$\phi_j(x) = x^j.$$



Basis functions are global: small changes in **x** affect all basis functions.

Gaussian basis functions:

$$\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right).$$

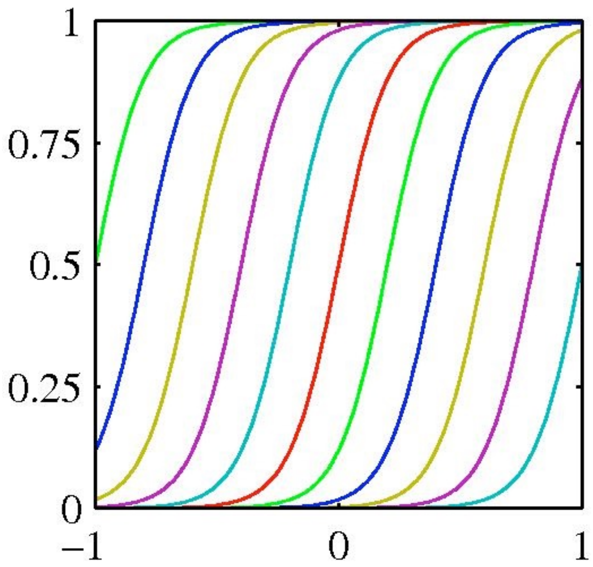


Basis functions are local: small changes in **x** only affect nearby basis functions.
μ_j and s control location and scale (width).

Linear Basis Function Models

Sigmoidal basis functions

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right), \text{ where } \sigma(a) = \frac{1}{1+\exp(-a)}.$$



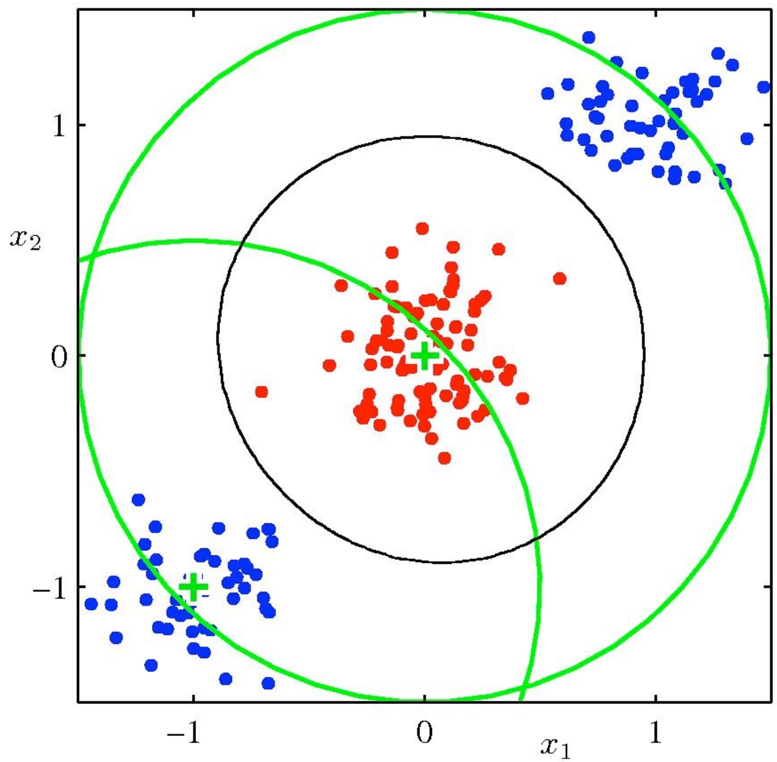
Basis functions are local: small changes in **x** only affect nearby basis functions. μ_j and s control location and scale (slope).

• Decision boundaries will be linear in the feature space ϕ, but would correspond to nonlinear boundaries in the original input space x.

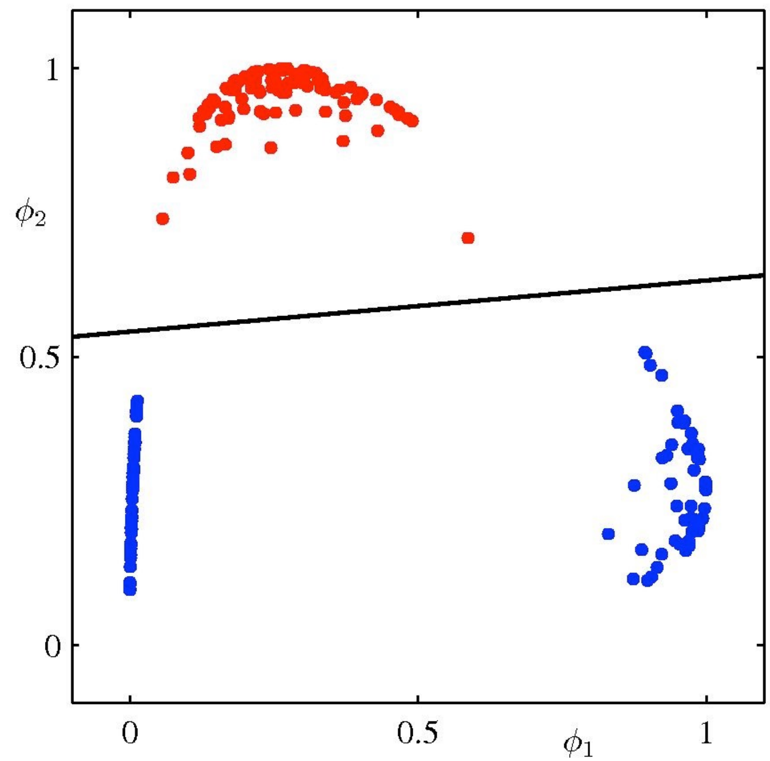
• Classes that are linearly separable in the feature space $\phi(\mathbf{x})$ need not be linearly separable in the original input space.

Linear Basis Function Models

Original input space



Corresponding feature space using two Gaussian basis functions



- We define two Gaussian basis functions with centers shown by green the crosses, and with contours shown by the green circles.

- Linear decision boundary (right) is obtained using logistic regression, and corresponds to nonlinear decision boundary in the input space (left, black curve).

Maximum Likelihood

- As before, assume observations arise from a deterministic function with an additive Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

which we can write as:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

- Given observed inputs $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, and corresponding target values $\mathbf{t} = [t_1, t_2, ..., t_N]^T$, under i.i.d assumption, we can write down the likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta),$$

where $\boldsymbol{\phi}(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), ..., \phi_{M-1}(\mathbf{x}))^T$.

Maximum Likelihood

Taking the logarithm, we obtain:

$$\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \sum_{i=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta)$$

$$= -\frac{\beta}{2} \underbrace{\sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\right)^2}_{\text{sum-of-squares error function}} + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

sum-of-squares error function

Differentiating and setting to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

Maximum Likelihood

Differentiating and setting to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

Solving for **w**, we get:

$$\mathbf{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t}$$

The Moore-Penrose pseudo-inverse, $\boldsymbol{\Phi}^{\dagger}$.

where $\boldsymbol{\Phi}$ is known as the design matrix:

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

Sequential Learning

• The training data examples are presented one at a time, and the model parameters are updated after each such presentation (online learning):

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \bigtriangledown E_n$$

weights after seeing training case t+1

learning rate

vector of derivatives of the squared error w.r.t. the weights on the training case presented at time t.

• For the case of sum-of-squares error function, we obtain:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left(t_n - \mathbf{w}^{(t)^T} \boldsymbol{\phi}(\mathbf{x}_n) \right) \boldsymbol{\phi}(\mathbf{x}_n).$$

• Stochastic gradient descent: The training examples are picked at random (dominant technique when learning with very large datasets).

• Care must be taken when choosing learning rate to ensure convergence.

Regularized Least Squares

- Let us consider the following error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

λ is called the regularization coefficient.

- Using sum-of-squares error function with a quadratic penalization term, we obtain:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

which is minimized by setting:

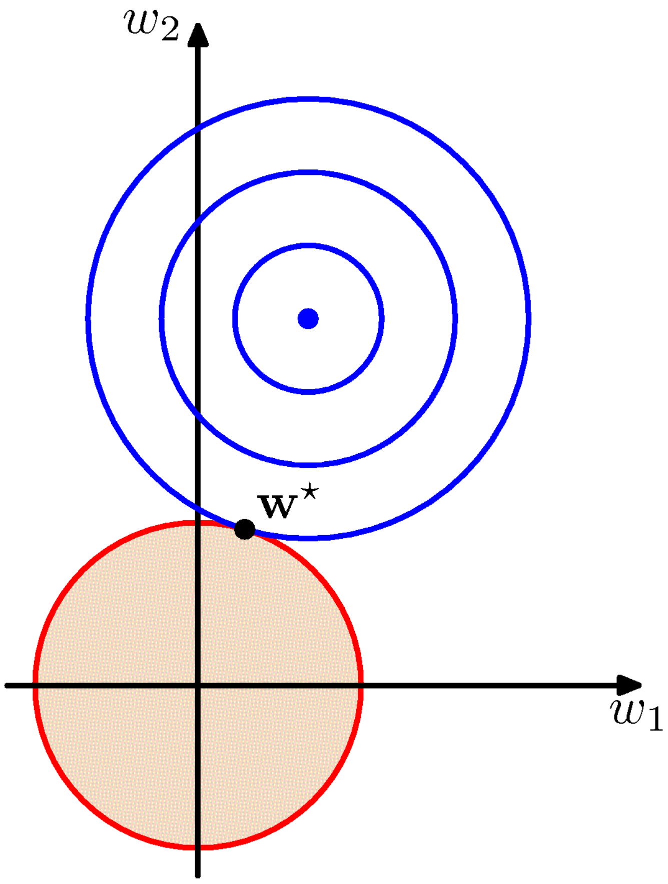
Ridge regression

$$\mathbf{w} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$

The solution adds a positive constant to the diagonal of $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$. This makes the problem nonsingular, even if $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$ is not of full rank (e.g. when the number of training examples is less than the number of basis functions).

Effect of Regularization

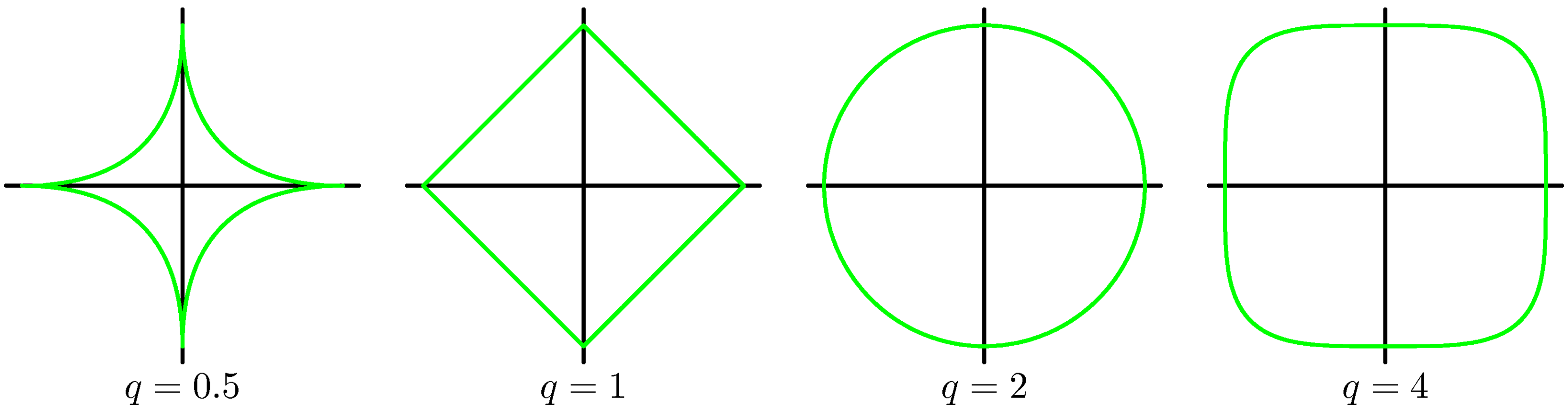
- The overall error function is the sum of two parabolic bowls.
- The combined minimum lies on the line between the minimum of the squared error and the origin.
- The regularizer shrinks model parameters to zero.



Other Regularizers

Using a more general regularizer, we get:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



Lasso

Quadratic

The Lasso

- Penalize the absolute value of the weights:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{M-1} |w_j| \right].$$

- For sufficiently large λ, some of the coefficients will be driven to exactly zero, leading to a sparse model.

- The above formulation is equivalent to:

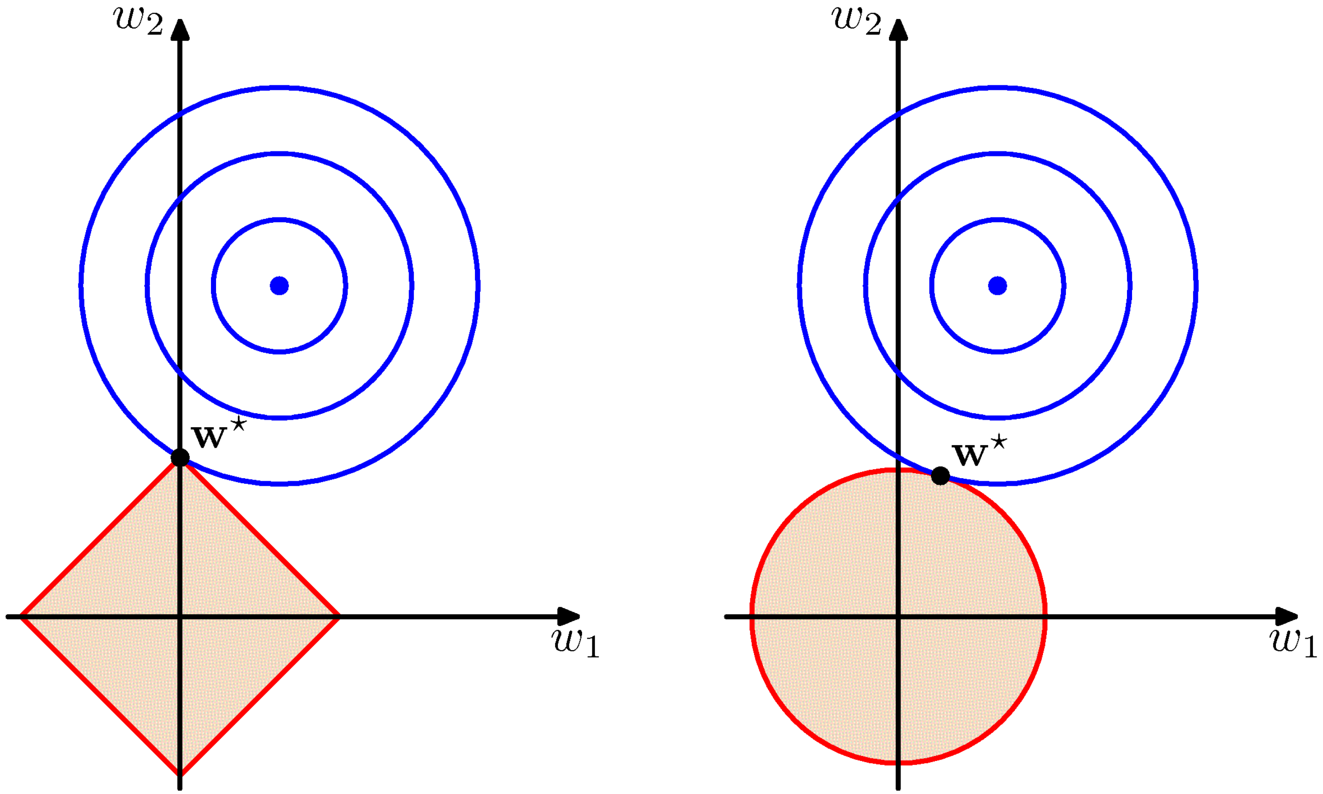
$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \; \underbrace{\frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \right)^2}_{\text{unregularized sum-of-squares error}}, \text{ subject to } \sum_{j=1}^{M-1} |w_j| \leq \tau.$$

- The two approaches are related using Lagrange multiplies.

- The Lasso solution is a quadratic programming problem: can be solved efficiently.

Lasso vs. Quadratic Penalty

Lasso tends to generate sparser solutions compared to a quadratic regualrizer (sometimes called L_1 and L_2 regularizers).



Bias-Variance Decomposition

- Introducing a regularization term can help us control overfitting. But how can we determine a suitable value of the regularization coefficient?

- Let us examine the expected squared loss function. Remember:

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$

for which the optimal prediction is given by the conditional expectation:

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x})\,\mathrm{d}t.$$

intrinsic variability of the target values: The minimum achievable value of expected loss

- If we model $h(\mathbf{x})$ using a parametric function $y(\mathbf{x}, \mathbf{w})$, then from a Bayesian perspective, the uncertainly in our model is expressed through the posterior distribution over parameters **w**.

- We first look at the frequentist perspective.

Bias-Variance Decomposition

- From a frequentist perspective: we make a point estimate of $\mathbf{w}^*$ based on the dataset D.

- We next interpret the uncertainly of this estimate through the following thought experiment:

  - Suppose we had a large number of datasets, each of size N, where each dataset is drawn independently from $p(\mathbf{x}, t)$.
  - For each dataset D, we can obtain a prediction function $y(\mathbf{x}; \mathcal{D})$.
  - Different datasets will give different prediction functions.
  - The performance of a particular learning algorithm is then assessed by taking the average over the ensemble of these datasets.

- Let us consider the expression:

$$\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2.$$

- Note that this quantity depends on a particular dataset D.

Bias-Variance Decomposition

- Consider:

$$\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2.$$

- Adding and subtracting the term $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]$, we obtain

$$\begin{aligned}
&\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2 \\
&= \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 \\
&= \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 \\
&\quad +2\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}.
\end{aligned}$$

- Taking the expectation over $\mathcal{D}$, the last term vanishes, so we get:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2\right] \\
&= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right]}_{\text{variance}}.
\end{aligned}$$

Bias-Variance Trade-off

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

Average predictions over all datasets differ from the optimal regression function.

Solutions for individual datasets vary around their averages -- how sensitive is the function to the particular choice of the dataset.

Intrinsic variability of the target values.

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

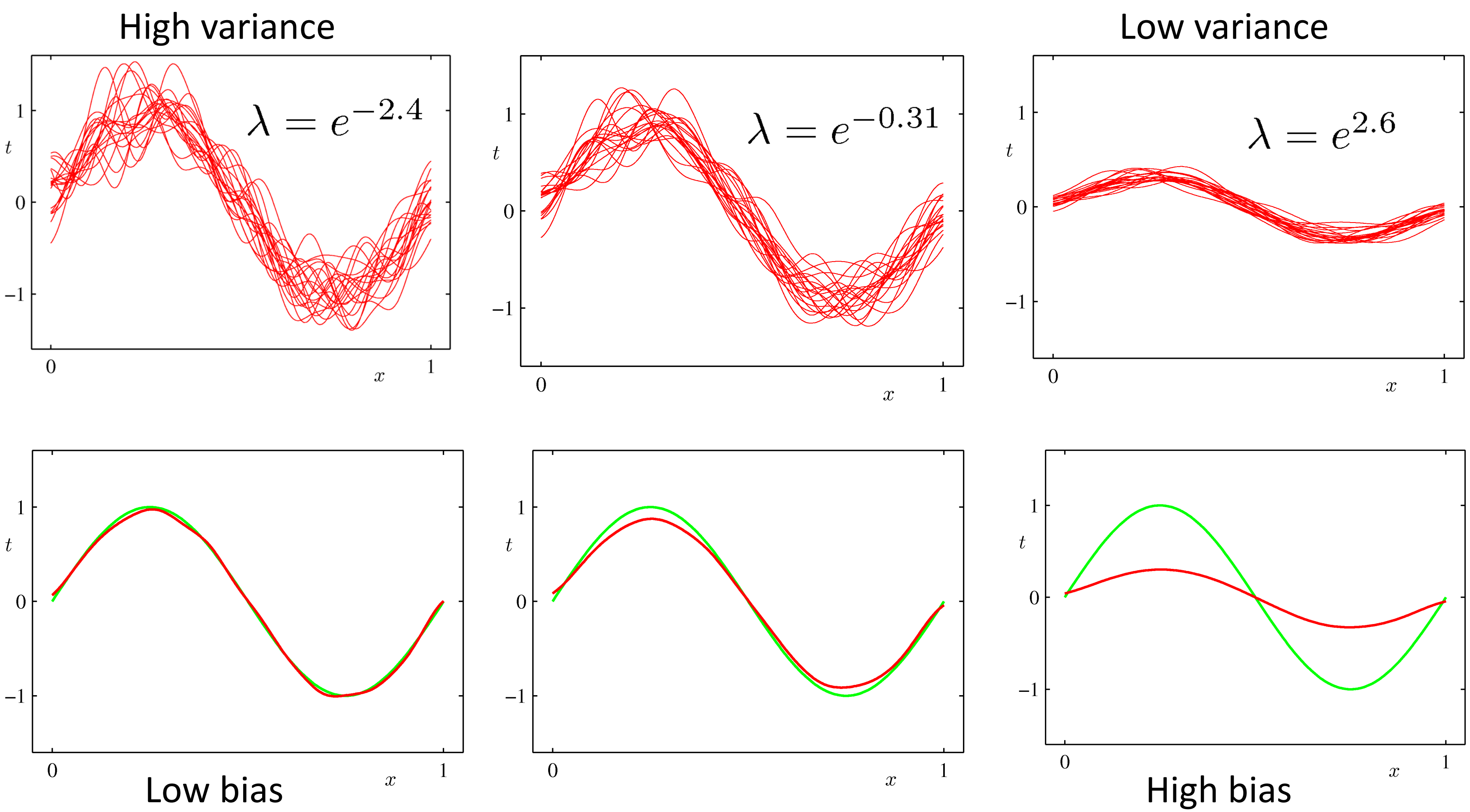
$$\text{variance} = \int \mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

$$\text{noise} = \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$

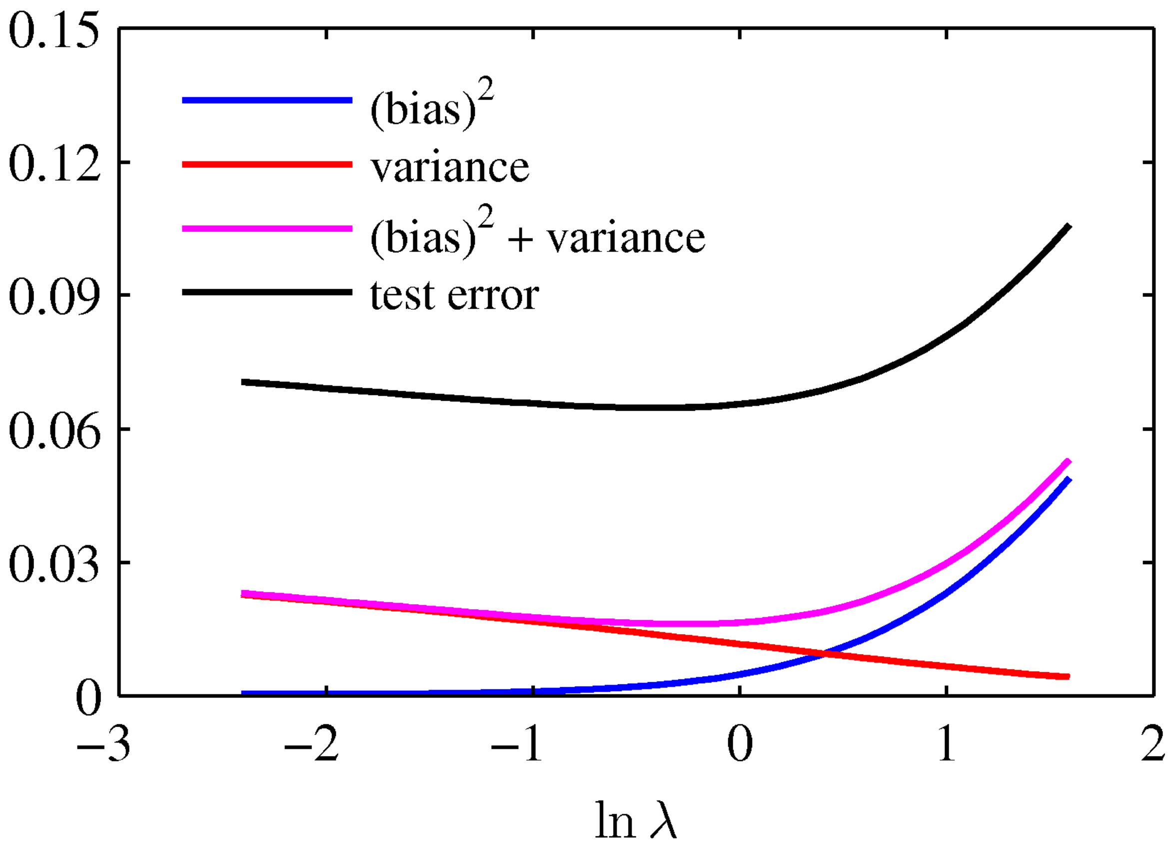
- Trade-off between bias and variance: With very flexible models (high complexity) we have low bias and high variance; With relatively rigid models (low complexity) we have high bias and low variance.
- The model with the optimal predictive capabilities has to balance between bias and variance.

Bias-Variance Trade-off

- Consider the sinusoidal dataset. We generate 100 datasets, each containing N=25 points, drawn independently from $h(x) = \sin 2\pi x$.



Bias-Variance Trade-off



From these plots note that over-regularized model (large λ) has high bias, and under-regularized model (low λ) has high variance.

Beating the Bias-Variance Trade-off

- We can reduce the variance by averaging over many models trained on different datasets:
  - In practice, we only have a single observed dataset. If we had many independent training sets, we would be better off combining them into one large training dataset. With more data, we have less variance.
- Given a standard training set D of size N, we could generate new training sets, N, by sampling examples from D uniformly and with replacement.
  - This is called bagging and it works quite well in practice.
- Given enough computation, we could also resort to the Bayesian framework:
  - Combine the predictions of many models using the posterior probability of each parameter vector as the combination weight.